Modellierung und Techniken zur Optimierung von Multiagentensystemen in Zellularen Automaten

Modeling and Optimization Techniques for Multi Agent Systems in Cellular Automata

Vom Fachbereich Informatik der Technischen Universität Darmstadt zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.) genehmigte **Dissertation** von Dipl.-Inform. Patrick Ediger, geboren am 17. April 1980 in Bad Soden

TECHNISCHE UNIVERSITÄT DARMSTADT

D17

Darmstadt, 2011

Referenten: Prof. Dr. Rolf Hoffmann
Prof. Dr. Dietmar Fey
Prof. Dr. Johannes Fürnkranz

Tag der Einreichung: 4. April 2011
Tag der Prüfung: 1. Juni 2011

Modellierung und Techniken zur Optimierung von Multiagentensystemen in Zellularen Automaten
Modeling and Optimization Techniques for Multi Agent Systems in Cellular Automata

genehmigte **Dissertation** von Dipl.-Inform. Patrick Ediger, geboren am 17. April 1980 in Bad Soden

1. Gutachten: Prof. Dr. Rolf Hoffmann
2. Gutachten: Prof. Dr. Dietmar Fey
3. Gutachten: Prof. Dr. Johannes Fürnkranz

Tag der Einreichung: 4. April 2011
Tag der Prüfung: 1. Juni 2011

Darmstadt — D 17

Sois un grandísimo bellaco -dijo a esta sazón don Quijote-; y vos sois el vacío y el menguado, que yo estoy más lleno que jamás lo estuvo la muy hideputa puta que os parió.

- Miguel de Cervantes Saavedra

Vorwort

Viereinhalb Jahre dauerte die Erstellung dieser Arbeit. In dieser Zeit haben viele Menschen etwas zum Gelingen meines Promotionsvorhabens beigetragen. Der Doktor ist der höchste akademische Grad, und insofern ein guter Augenblick, Bilanz zu ziehen und mich bei diesen Menschen zu bedanken. Die Reihenfolge der im Folgenden genannten Personen und Gründe spiegelt nicht unbedingt den Grad der Dankbarkeit wieder.

Zuallererst danke ich besonders meinem Doktorvater Professor Rolf Hoffmann. Dafür, dass er mir die Gelegenheit gab, bei ihm zu promovieren, für viele Ratschläge und anregende Diskussionen. In Zusammenarbeit entstand eine Vielzahl von Veröffentlichungen, die inhaltlich wesentlich für diese Arbeit sind. Außerdem möchte ich mich für das angenehme Arbeitsverhältnis, in dem er mir viele Freiheiten ließ und sich als guter und motivierender Chef erwiesen hat, bedanken. Ohne Professor Hoffmann hätte es keine Dissertation von mir gegeben.

Ich bedanke mich auch bei Professor Karsten Weihe, der mich als Dekan des Fachbereichs nicht nur für die Promotion zugelassen hat, sondern mir auch dadurch eine Finanzierung ermöglicht hat, dass er mich als Koordinator für das Mentorenprogramm für Studienanfänger eingesetzt hat.

Neben Professor Hoffmann haben zwei weitere Referenten meine Arbeit begutachtet. Dafür bedanke ich mich bei Professor Dietmar Fey aus Erlangen und Professor Johannes Fürnkranz aus Darmstadt. Weiterhin zu Dank verpflichtet bin ich Professor Michael Gösele und Professor Thorsten Strufe, die als Prüfer in meiner Disputation mitgewirkt haben, sowie Professor Max Mühlhäuser, der die Disputation als Vorsitzender geleitet hat.

Ich danke Marcus Komann für das Korrekturlesen meiner Arbeit und die tolle Zusammenarbeit bei Veröffentlichungen. Ich danke weiteren Kollegen ebenfalls für eine gute und nützliche Zusammenarbeit bei Veröffentlichungen: Dietmar Fey aus Erlangen, Dominique Désérable aus Rennes, Johannes Jendrszcok, Sylvia Grüner und Mathias Halbach. Letzterer hat außerdem durch seine Dissertation eine wichtige inhaltliche Grundlage für meine Arbeiten gelegt. Auch dafür sei ihm gedankt.

Bei meinen Kollegen Johannes Jendrszcok, Christian Schäck, Thomas Paul, Wolfgang Heenes und Gudrun Harris bedanke ich mich für die tolle Atmosphäre im Büro und viele hilfreiche Tipps und anregende Diskussionen in technischen sowie inhaltlichen Fragen, aber auch für die manchmal notwendige Ablenkung. Wolfgang Heenes diente mir außerdem insbesondere bei allen methodischen Fragen und Fragen zum wissenschaftlichen Arbeiten als Vorbild und Ratgeber. Gudrun Harris war in allen organisatorischen Fragen eine unersetzbare Hilfe.

Auch außerhalb des Büros – die Doktorarbeit verschwand leider auch nach Feierabend nicht immer aus dem Kopf – bin ich von vielen unterstützt worden. Mein Dank gilt hier meiner Familie und meinen Freunden. Bei meinen Eltern bedanke ich mich auch für die finanzielle Unterstützung während meines gesamten Studiums.

Darmstadt, Juni 2011
Patrick Ediger

Kurzfassung

Für eine Reihe von Problemstellungen werden heutzutage Multiagentensysteme als dezentral organisierte Systeme mit autonom und parallel agierenden Agenten eingesetzt. Solche Systeme sollen oft Eigenschaften wie Adaptivität, Selbstheilung, Kontrollierbarkeit oder Ausfallsicherheit aufweisen.

Schlagworte wie Schwarmintelligenz, Emergenz und Synergie werden verwendet, um zu beschreiben, wie aus möglicherweise einfachen lokalen Subsystemen komplexe globale Systeme entstehen. Der systematische Entwurf von lokal agierenden Agenten, so dass eine globale Aufgabe erfüllt werden kann, ist nicht trivial. Aber durch die Parallelität der Verarbeitung und die relative Einfachheit der einzelnen Agenten kann ein Einsatz von Multiagentensystemen im Vergleich zu einer komplexen Verarbeitungseinheit zu einer Verbesserung der Leistung und/oder zu einer Verringerung des Ressourcenbedarfs führen. Z. B. können in Kamerachips Multiagentensysteme zur parallelen Verarbeitung von Bilddaten eingesetzt werden, oder zu adaptiven Routing-Algorithmen in Netzwerken auf Chips. Der schwierige Prozess des Entwurfs eines emergenten Systems kann sich daher trotzdem lohnen.

Das generelle Ziel dieser Arbeit ist es, erstens ein Modell zur Beschreibung von Multiagentensystemen zu entwickeln, das für beliebige Strukturen der Agentenkontrolle (Verhaltenssteuerung) geeignet ist und zweitens Techniken zur Optimierung dieser Strukturen zu entwerfen.

Zellulare Automaten können als Modell für Multiagentensysteme, denen eine räumliche diskretisierbare Struktur anhaftet, und die in Hardware-Chips implementiert werden sollen, gut verwendet werden. In dieser Arbeit wird ein allgemeines Modell für Multiagentensysteme in Zellularen Automaten entworfen, das mit einer beliebigen Struktur für die Programmierung des Agentenverhaltens versehen werden kann.

Es werden für drei Beispielanwendungen Multiagentensysteme in dem entworfenen Modell Agenten mit einer Steuerung durch endliche Zustandsautomaten eingesetzt: das vollständige Erkunden eines in kleine Segmente aufgeteiltes, unbekanntes Gebiet mit Hindernissen, die vollständige Verbreitung von initial, gegenseitig exklusiv verteilter Information und der Transport von Nachrichten in einem zweidimensionalen Mesh-Netzwerk. Das Agentenverhalten, also der Inhalt der endlichen Zustandsautomaten, wird dabei mit Genetischen Algorithmen heuristisch optimiert, um die vorher definierten, globalen Ziele zu erreichen. Es werden außerdem generelle Optimierungstechniken für den strukturellen Aufbau der Agenten entworfen und Maßnahmen zur Anpassung der Konfiguration des Zellularen Automaten vorgestellt, alle mit dem Ziel, das Multiagentensystem als Ganzes effektiver und effizienter zu machen.

Am Schluss der Arbeit werden Experimente und Ergebnisse präsentiert, die die Effektivität und Effizienz der Genetischen Algorithmen, sowie der Optimierungstechniken für die Agentenstruktur bewerten. Die Genetischen Algorithmen bewähren sich in allen Beispielsystemen und finden akzeptable Agentenverhalten. Nicht alle Anpassungen in der Agentenstruktur und der Konfiguration des Zellularen Automaten sind effektiv und effizient, aber erfolgreich sind insbesondere solche Optimierungen, die eine räumliche oder zeitliche Heterogenität oder eine Randomisierung in das Verhalten der Agenten hineinbringen.

Abstract

Nowadays Multi Agent Systems are applied for many problems as decentrally organized systems with agents acting autonomously and in parallel. In many cases it is desired to find capabilities like adaptivity, self-healing, controllability and reliability in such systems.

The words swarm intelligence, emergence and synergy are often used to describe the phenomenon of the development of a complex global system behavior with only locally acting and possibly simple subsystems. A systematic approach of designing local agents in a way such that a global goal is reached by guaranteeing reliability, controllability etc. at the same time, is not a trivial task. But the implementation of a Multi Agent System can still be an improvement compared to a single complex processing unit, if, due to the exploitation of parallelism and the simplicity of the agents, the performance can be increased and/or the need of resources can be reduced. E. g., Multi Agent Systems can be implemented for image processing in camera sensor chips or for adaptive routing algorithms in networks on a chip. Thus, in spite of the difficulties of the designing process of an emergent behavior, it might be worth doing it.

The general goal of this work is on one hand the development of a Multi Agent System model, which is suitable for an arbitrary agent control structure (agent behavior control) and on the other hand the design of techniques for the optimization of the control structure.

Cellular Automata are generally well applicable as a model for Multi Agent Systems with a spatially discrete structure and which are supposed to be implemented in hardware chips. In this thesis, a general model for Multi Agent Systems in Cellular Automata is developed, which can incorporate an arbitrary structure for the programming of the agent behavior.

Three example applications of Multi Agent Systems are defined in this thesis: the complete exploration of an unknown area with obstacles, that is divided in small sections, the complete distribution of initially mutually exclusively distributed information and the transport (routing) of messages in a two-dimensional mesh network. Agents that are controlled by a finite state machine are used in these examples. The behavior of the agents (i. e., the content of the finite state machines) is evolved with Genetic Algorithms, in order to reach the previously defined goals. Additionally, optimization techniques for the structural assembly of the agents are designed. Furthermore, methods for the adjustment of the configuration of the Cellular Automaton are presented. All of the methods and techniques are suggested with the intention to increase the effectivity and the efficiency of the Multi Agent Systems as a whole.

Eventually, experiments and results are presented, which evaluate the effectivity and efficiency of the Genetic Algorithms, the optimization techniques for the agents' structure as well as the methods for the adjustment of the Cellular Automaton. The Genetic Algorithms perform well in any of the example systems and find feasible solutions (agent behaviors). Not all of the adjustments and optimization techniques are effective and efficient. Particularly those methods that induce a spatial or temporal heterogeneity or that use a certain amount of randomness in the agents' behavior, are effective.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Technische Systeme mit verteilten, autonom agierenden Agenten zur Lösung komplexer Probleme finden heutzutage in Forschung und Industrie unter den Begriffen *verteilte künstliche Intelligenz* (DAI, engl.: distributed artificial intelligence) und *Multiagentensysteme* (MAS) breite Verwendung. Einige der bekanntesten Anwendungen sind Computerspiele, autonome Roboter, Verkehrssimulationen, Optimierung von Produktionsprozessen oder Transportsystemen, sowie die Erforschung von sozialem Verhalten in Gemeinschaften.

Zwei grundsätzlich unterschiedliche Anwendungsformen von MAS können unterschieden werden. In der ersten geht es darum, ein in der Natur oder Gesellschaft real vorkommendes System möglichst originalgetreu nachzubilden, um durch Simulation das Gesamtverhalten des Systems zu untersuchen, um Verhaltensweisen zu erklären oder um Vorhersagen zu machen. In der zweiten Anwendungsform wird ein künstliches, mathematisches oder abstraktes System benutzt. Dabei ist entweder ein Problem gegeben, das von den Agenten möglichst gut gelöst werden soll, d. h. ein dementsprechendes Verhalten der Agenten wird gesucht, oder es ist ein Agentenverhalten vorgegeben und die daraus global entstehenden Phänomene werden untersucht. Im Vordergrund dieser Arbeit steht die zweite Form der Anwendung, in der autonome Agenten ein Problem möglichst gut lösen sollen. Ein besonderer Aspekt dabei ist die *Schwarmintelligenz*, eine Eigenschaft von Systemen mit vielen kleinen autonomen Einheiten, die in ihrer Gesamtheit ein intelligentes Verhalten haben. Der Aspekt der technischen Realisierung von künstlicher Schwarmintelligenz ist sogar soweit in das öffentliche Bewusstsein vorgedrungen, dass er Einzug in die Unterhaltungsliteratur und andere Medien (z. B. Film) gefunden hat. Beispiele für die Verarbeitung der Thematik in der Fiktion sind [Cri02], [Ros95], [Lem67] und [Sch04], wobei in letzterem keine künstlichen, sondern natürliche (aber fiktive) MAS auftreten.

1.1 Motivation und Ziele

In der Natur kommen Multiagentensysteme vor, die ob der relativen Einfachheit der Agenten ein verblüffend komplexes Gesamtverhalten zeigen. Das Verhalten von Ameisenkolonien oder Bienenvölkern sind Beispiele hierfür [GGT07]. Ein technisches, von der Natur inspiriertes verteiltes System ohne zentrale Kontrolle zu entwerfen, das ein ebenso komplexes Gesamtverhalten zeigt, ist kein triviales Unterfangen. Dennoch kann es sinnvoll sein, ein dezentral kontrolliertes System zu entwerfen, wenn z. B. die Implementierungsplattform eine parallele Verarbeitung erlaubt und somit Geschwindigkeitsvorteile entstehen können. Weitere Vorteile könnten eine verbesserte Ausfallsicherheit oder Stabilität des Systems sein, da der Ausfall oder das fehlerhafte Verhalten einer einzelnen Komponente (bzw. eines einzelnen Agenten) nicht unbedingt kritisch für das Gesamtverhalten sein muss, im Gegensatz zu einem Ausfall der zentralen Kontrolle. Ein Beispiel für solche MAS mit *Selbstheilungseigenschaften* ist die Autonomous Nano Technology Swarm (ANTS) Mission der National Aeronautics and Space Administration (NASA), in dem unterschiedliche Agententypen unterschiedliche Aufgaben übernehmen und bei Bedarf ihre Aufgabe wechseln können [THRR04].

Ein Rechenmodell, das zur Modellierung von MAS häufig verwendet wird, ist das Modell des *Zellularen Automaten* (CA, engl.: Cellular Automata) [Gru09]. Der CA besteht aus einer großen Anzahl von Zellen, die durch eine statische Nachbarschaft verbunden sind. Die Nachbarschaft ist lokal, d. h. räumlich beschränkt. In zeitdiskreten Schritten berechnen alle Zellen parallel aus den Informationen ihres eigenen Zustands und denen der Zustände ihrer Nachbarn den nächsten eigenen Zustand. Dabei haben alle Zellen nur Lesezugriff auf ihre Nachbarn. Der CA ist also ein räumlich und zeitlich diskretes Modell. MAS mit ebensolchem Charakter und Agenten mit lokalem Interaktionsradius können also gut im CA-Modell beschrieben und simuliert werden. Das CA-Modell wird auch als massivparalleles Modell bezeichnet, da es über eine sehr große Anzahl von einfachen Berechnungseinheiten (Zellen) verfügen kann (im Gegensatz zu parallelen Multiprozessorsystemen mit nicht so vielen Prozessoren, aber dafür komplexeren Berechnungseinheiten).

Ein weiterer Grund, der für die Modellierung von MAS im CA-Modell spricht, ist die einfache Realisierbarkeit in Hardware. Durch die synchrone Generationsberechnung und die Vermeidung von Schreibzugriffen auf Nachbarzellen, können leistungsfähige Hardwarearchitekturen entwickelt werden. Eine Anwendung von solchen Hardwarearchitekturen für MAS in CA sind eingebettete Systeme, wie z. B. in Kameras [FS05]. Das CA-Modell scheint daher insgesamt für solch ein Multiagentensystem geeignet zu sein.

Das Hauptziel dieser Arbeit ist es, ein allgemeines Modell für MAS in CA zu entwickeln und dafür geeignete generelle Techniken zu entwickeln, die es ermöglichen, eine Verhaltensvorschrift für autonome Agenten in *dezentral kontrollierten Systemen* zu finden, so dass die Agenten ein vorgegebenes globales Problem möglichst effizient lösen. Dieses Hauptziel lässt sich in zwei Teilziele aufspalten:

a) **Modellentwurf.** Das erste Teilziel ist der *Entwurf eines MAS-Modells zur Beschreibung von Multiagentensystemen im Modell des Zellularen Automaten*. Dazu gehört die Modellierung der Umwelt sowie die Struktur der Agenten, inklusive einer Definition des Begriffs Agent im Zusammenhang mit Zellularautomaten. Das Modell soll so gestaltet sein, dass das grundlegende Prinzip anwendbar für unterschiedliche Systeme mit verschiedenen Agententypen, unterschiedlichen Eigenschaften der Umwelt und unterschiedlichen globalen Zielen ist. Das Modell soll weiterhin für eine effiziente Realisierung in speziellen Hardwarearchitekturen geeignet sein, und so der inhärent massivparallelen Natur des Zellularautomaten gerecht werden.

b) **Techniken zur Verhaltensoptimierung.** Zwei Aspekte dieses Teilziels sind zu unterscheiden. Es soll erstens *ein geeignetes prinzipielles Verfahren entwickelt werden, mit dem das Agentenverhalten heuristisch optimiert werden kann*. Dieses Verfahren soll es ermöglichen, lokales Agentenverhalten zu finden, welche deren konkrete globale Aufgabe möglichst optimal löst. Außerdem sollen *Techniken entwickelt werden, die es erlauben, die Agentenstruktur und das darauf prinzipiell anwendbare heuristische Verfahren zur Optimierung des Agentenverhaltens problemspezifisch anzupassen*. Die Eignung dieser angepassten Verfahren soll anhand von geeigneten Multiagentenproblemen (Multiagentensysteme mit einem spezifischen globalen Ziel) überprüft werden.

Die Modellierung der Agentenstruktur muss für die zu entwickelnden Heuristiken geeignet sein. Oder umgekehrt formuliert, muss die Heuristik auf die Agentenstruktur anwendbar sein. Außerdem muss sie flexibel in dem Sinne sein, dass es damit möglich ist, für ein beliebiges glo-

bales Problem ein Verhalten zu erzeugen, das das Problem löst. Anpassungen der Agentenstruktur sind dabei mit eingeschlossen. Eine Anpassung kann in mehrerlei Hinsicht geschehen, z. B. durch die manuelle Gestaltung der Struktur des zu optimierenden Agentenverhaltens, durch die Anpassung oder Erweiterung der Fähigkeiten der Agenten, d. h. der möglichen Aktionen und der detektierten Eingangssignale, durch die Veränderung der Parameter der Heuristik und der Qualitätsbewertung des Agentenverhaltens oder durch den verteilten Einsatz von Heuristiken auf Teilprobleme und geeignete Zusammenführung der Teillösungen.

Insgesamt soll der Fokus nicht auf der Optimierung der Heuristik selbst, sondern auf der Modellierung der Struktur der Agenten liegen. Die bei der Optimierung verwendete Heuristik ist eher als notwendiges Instrument zu sehen, welches natürlich im Rahmen des zweiten Teilziels trotzdem entsprechend gestaltet werden muss, so dass es auf das Modell passt und möglichst gute Ergebnisse liefert.

1.2 Struktur der Arbeit

Im folgenden Kap. 2 wird die Verwendung des Begriffes *Agent* in der Literatur diskutiert. Damit einhergehend werden Eigenschaften von Agenten und Multiagentensystemen, die zur Klassifizierung verwendet werden, benannt und beschrieben. Es wird dann eine Reihe von Möglichkeiten zur Anwendung von Multiagentensystemen aufgezeigt, die sich aus ihren besonderen Eigenschaften ergeben.

Im gleichen Kapitel werden die Grundlagen sowie einige typische Anwendungen des Modells des Zellularen Automaten vorgestellt. Schließlich werden Multiagentensysteme aus der Literatur besprochen, die mit Zellularen Automaten modelliert wurden. Dabei werden die grundsätzlichen Möglichkeiten zur Interpretation des Zusammenhangs von Zellularen Automaten und Multiagentensystemen aufgezeigt.

In Kap. 3 wird ein allgemeines Modell für Multiagentensysteme im Zellularen Automaten entwickelt. Dazu wird die Struktur der Zellen im CA genau betrachtet und so gestaltet, dass beliebige MAS damit modelliert werden können. Das Modell abstrahiert von der Agentenstruktur insoweit, dass darin verschiedene Möglichkeiten zur Kontrolle des Agenten realisiert werden können. Einige Möglichkeiten, das Verhalten der Agenten zu implementieren und in das Modell zu integrieren, werden untersucht und eine für den Zweck der Untersuchung von Optimierungsverfahren passende Variante ausgewählt.

Für die Untersuchung von Optimierungsverfahren für Agentenverhalten werden bestimmte Beispielanwendungen von Multiagentensystemen in Zellularen Automaten ausgewählt. Diese Anwendungen werden im Detail in Kap. 4 beschrieben.

Kap. 5 beschäftigt sich mit der Konstruktion und der Optimierung des Verhaltens der Agenten. Zunächst wird auf grundsätzlich bei der Optimierung zu berücksichtigende Eigenschaften eines MAS eingegangen und dann heuristische Verfahren vorgestellt und deren Anwendbarkeit auf die Optimierung des in Kap. 3 entwickelten Agentenverhaltensmodells diskutiert. Anhand von Beispielsystemen wird die Effektivität eines prinzipiellen heuristischen Verfahrens untersucht.

Dann werden Techniken entwickelt, die auf der manuellen Gestaltung der Struktur des Agentenverhaltens beruhen und kombiniert mit den Heuristiken eingesetzt werden sollen.

In Kap. 6 werden die Techniken der manuellen Modellierung und die heuristischen Verfahren kombiniert auf bestimmte MAS angewendet. Der Aufbau dieser Experimente wird beschrieben

und die Resultate vorgestellt und ausgewertet. Damit sollen die vorher entwickelten Techniken validiert werden.

Kap. 7 enthält eine Zusammenfassung der Ergebnisse der Arbeit und Schlussfolgerungen im Bezug auf die vorgegebenen Ziele und gibt einen Ausblick auf zukünftige Fragestellungen.

Im Anhang befinden sich Quelltexte, verschiedene Abbildungen von Konfigurationen von Multiagentensystemen, die als Testbeispiele verwendet worden sind, Listen von evolvierten endlichen Zustandsautomaten, die ein interessantes Agentenverhalten auslösen, und ein Glossar, in dem insbesondere die bei der Modellierung in Kap. 3 verwendeten Begriffe erklärt sind.

2 Grundlagen von Multiagentensystemen und Zellularen Automaten

Multiagentensysteme sind in der Forschung schon seit etwa drei Jahrzehnten ein aktuelles Forschungsthema [Wei99, S. 1], und seit Mitte der 90er Jahre ein weit verbreitetes und oft verwendetes Konzept [Woo02, S. xi]. Ursprünglich aus dem Gebiet der verteilten künstlichen Intelligenz hervorgehend, gibt es heute viele verschiedene Einsatzgebiete [BL04, Wei99, S. 6-7], bei denen auch das Verständnis des Begriffs *Agent* stark unterschiedlich ist. Dieses Kapitel liefert eine Definition des Begriffs Agent, so wie er in dieser Arbeit verwendet wird. Außerdem werden unterschiedliche Eigenschaften von Agenten und Multiagentensystemen beschrieben, und es gibt einen Überblick über eine Auswahl von populären Anwendungen von Multiagentensystemen in der Literatur.

In dieser Arbeit werden MAS mit Hilfe von Zellularen Automaten (CA) modelliert, weil das CA-Modell aufgrund seiner parallelen Verarbeitungsmöglichkeit und den gut auf Agentensystemen mit lokal agierenden Agenten übertragbaren Eigenschaften dafür geeignet erscheint. Das Modell wird in diesem Kapitel vorgestellt und der Zusammenhang von CA und MAS aufgezeigt, sowie die besonderen Eigenschaften von MAS in CA. Schließlich werden konkrete Beispiele für Multiagentensysteme in CA vorgestellt.

2.1 Grundlagen von Multiagentensystemen

2.1.1 Definitionen und Charakteristika von Agenten

Zunächst ist es notwendig zu klären, was im Sinne dieser Arbeit überhaupt ein Agent ist. Etymologisch leitet sich der Begriff vom lateinischen agens (der Treibende) ab. Ohne einen speziellen Kontext zu kennen, ist ein Agent ein Vertreter, ein Vermittler oder ein Beauftragter [Wah00]. Im allgemeinen Sprachgebrauch sind verschiedene Bedeutungen üblich, die sich dann aus dem Kontext ergeben, aber alle mindestens einer der drei eben genannten Definitionen zugeordnet werden können, z. B. kann Agent eine der folgenden Bedeutungen haben: Geheimagent, Versicherungsagent, Schauspielagent, Literaturagent, etc. In [Mer10] findet sich unter anderem die dazu passende Definition für die englische Übersetzung agent:

„*one who is authorized to act for or in the place of another.*“

Mit Bezug auf den Themenkomplex der künstlichen Intelligenz gibt es zahlreiche verschiedene Definitionen, die einhergehen mit der Nennung bestimmter Eigenschaften von Agenten. Oft werden auch die Begriffe *intelligente Agenten* oder *autonome Agenten* gebraucht, um allgemeine Agenten von solchen zu unterscheiden, die selbst entscheiden können, welche Aktionen in gegebenen Situationen erforderlich sind. Nachfolgend werden einige der Definitionen von Agenten und intelligenten Agenten aus der Literatur zusammengefasst und die dort genannten

Eigenschaften erörtert. In Tab. 2.1 sind alle genannten Attribute zur Klassifizierung von Agenten mit kurzer Beschreibung und den in der Literatur dafür üblicherweise verwendeten englischen Begriffen aufgelistet. Zudem wird eine Zuordnung zu Quellen, in denen die besagte Eigenschaft für intelligente Agenten gefordert wird, gegeben.

Tab. 2.1.: Klassifikation von Agenten durch ihre Attribute.

Attribut	**engl. Begriff**	**kurze Beschreibung**	**gefordert bei**
autonom, *unabhängig*	autonomous, independent, self-directed	Der Agent kann ohne direktes Eingreifen von außen agieren.	[WJ95, Woo02, FG96, RN04, HR95, MN05, MN06]
wahrnehmend, *perzeptiv*	perceptive	Der Agent nimmt seine Umwelt wahr.	
reaktiv	reactive	Der Agent nimmt seine Umwelt wahr und reagiert darauf in einem zeitlich angemessenen Rahmen.	
pro-aktiv, *zielorientiert*	pro-active, goal-oriented, purposeful	Der Agent ergreift selbst die Initiative, um sein Ziel zu erreichen.	[WJ95, Woo02, FG96, MN05, MN06]
kommunikativ, *sozial fähig*	communicative, socially able	Agenten sind in der Lage, mit anderen Agenten zu interagieren.	[WJ95, Woo02]
deliberativ	deliberative	Mit dem Wissen über seine Umwelt kann der Agent einen Verhaltensplan entwickeln.	-
kontinuierlich	continuous	Agenten sind ein kontinuierlich laufender Prozess.	[FG96, HR95]
abgeschlossen, *identifizierbar*	self-contained, identifiable	Ein Agent ist ein diskretes Individuum und über seine Bestandteile eindeutig identifizierbar.	[MN05, MN06]
adaptiv, *lernend*	adaptive, learning	Agenten ändern ihr Verhalten aufgrund von gesammelten Erfahrungen.	[RN04, MN05, MN06]
beweglich, *mobil*	mobile	Agenten können sich von sich aus in ihrer Umwelt bewegen.	-
aufrichtig	honest	Agenten geben nicht absichtlich falsche Informationen weiter.	[GK94]
rational	rational	Agenten agieren nicht absichtlich so, dass ihre Ziele nicht erreicht werden können.	[RN04]
modifizierend	modifying	Agenten können den Zustand ihrer Umwelt verändern.	[HR95]
emotional, *antropomorph*	character	Der Agent besitzt eine glaubwürdige „Persönlichkeit“ und einen emotionalen Zustand.	-
dynamisch	dynamic	Agenten können sterben, geboren werden, sich vereinen oder sich klonen.	-

Wooldridge und Jennings geben in [WJ95] eine Definition von Agenten, die auf vier Eigenschaften beruht. Ein intelligenter Agent ist danach *autonom*, *reaktiv*, *pro-aktiv* und hat *soziale Fähigkeiten*. Hierbei bedeutet autonom, dass der Agent ohne direktes Eingreifen von außen in der Lage ist, Aktionen durchzuführen und dein eigenen internen Zustand zu kontrollieren. Ein indirektes Eingreifen, z. B. durch Veränderung der wahrgenommenen *Umwelt*, um gewisse Aktionen herbeizuführen, berührt nicht die Autonomität. Reaktive Agenten können in einem zeitlich an-

gemessenen Rahmen auf den von der Umwelt wahrgenommenen Zustand reagieren. Oft wird auch schon die Fähigkeit, die Umwelt nur wahrzunehmen als *perzeptive* Eigenschaft beschrieben. Pro-aktive Agenten ergreifen selbst die Initiative, um ein Ziel zu erreichen, anstatt nur zu reagieren. Es besteht also ein gewisser Zielkonflikt zwischen Reaktivität und Pro-Aktivität. Man könnte formulieren, je reaktiver ein Agent ist, desto weniger kann er selbst die Initiative ergreifen und pro-aktiv sein. Mit sozialen Fähigkeiten ist gemeint, dass ein Agent mit anderen Agenten in irgendeiner Form interagieren kann. In [Woo02, S. 15] formuliert Wooldridge zusammengefasst:

„*An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.*“

Dass der Agent sich in einer Umwelt befindet, wird auch häufig zu den Eigenschaften des Agenten gezählt („situatedness“). Über die Gestalt der Umwelt, in der sich der Agent befinden soll, wird hier keine genauere Angabe gemacht. Wooldridge's Definition ist der folgenden von Franklin und Graesser [FG96] nicht ganz unähnlich. Sie fassen mehrere Agentendefinitionen zusammen und geben dann eine eigene für autonome Agenten:

„*An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.*“

Ein Agent in diesem Sinne muss vier Attribute haben. Er muss wie auch bei Wooldridge und Jennings *reaktiv*, *autonom* und *pro-aktiv* sein. Außerdem wird gefordert, dass ein Agent ein *kontinuierlich* laufender Prozess sein soll. Das bedeutet, dass ein Agent für eine bestimmte Zeit lang agiert, ohne dass er neu dazu aufgefordert werden muss.

Eine Voraussetzung für Pro-Aktivität ist, dass der Agent über seine Entscheidungen nachdenken kann, d. h. er besitzt eine interne Repräsentation seiner Umwelt und kann mit diesem Wissen einen Plan entwickeln, um seine Ziele zu verfolgen [Mül96, S. 17]. Solche Agenten werden *deliberativ* genannt. Der Autor beschreibt dies anhand des Konzeptes der *BDI Agenten* (Belief Desire Intention). *Belief* ist die Annahme oder der „Glauben“, den die Agenten über den Zustand ihrer Umwelt haben. *Desire* beschreibt einen Zustand, der von den Agenten angestrebt wird. Mit *Intention* meint man die Absicht eines Agenten, einem konkreten Ziel (Desire) näher zu kommen.

Russel und Norvig definieren einen Agenten in [RN04, S. 55] so:

„*Ein Agent ist alles, was seine Umgebung über Sensoren wahrnehmen kann und in dieser Umgebung durch Aktuatoren handelt.*“

Wie schon in allen Definitionen zuvor, ist hier gefordert, dass ein Agent reaktiv sein soll. Autonomität und Pro-Aktivität sind zunächst nicht vorausgesetzt. Allerdings erfolgt nach dieser Definition eines allgemeinen Agenten, die Klassifizierung von weiteren Agententypen, welche als intelligent eingestuft werden. Zuerst führen die Autoren den *rationalen* Agenten ein, der sich immer so verhält, dass, gegeben seine Erfahrung und sein Wissen, die für seine Ziele optimale Aktion durchgeführt wird. Weiterhin fordern sie, dass ein intelligenter Agent auch *autonom* sein muss und ein *lernender* Agent sein muss, der sein *Verhalten* an sich verändernde Situationen oder Abläufe anpassen kann, nachdem er bestimmte Erfahrungen gesammelt hat. Als Verhalten wird die Gesamtheit aus Verarbeitung der Wahrnehmungen und Entscheidung für bestimmte Aktionen bezeichnet.

Hayes-Roth bietet folgende Definition [HR95]:

„Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions.“

Diese Agenten sind *perzeptiv* und *modifizierend*, da sie Aktionen durchführen, die den Zustand der Umwelt beeinflussen. Was Hayes-Roth unter reasoning versteht, ist das Bindeglied zwischen der Wahrnehmung und der Aktion des Agenten. Ihr Agent handelt also *reaktiv*, wobei pro-aktive Agenten nicht ausgeschlossen sind.

Obwohl keine zwei der genannten Definitionen identisch sind, gibt es doch eine weitgehende Übereinstimmung dahingehend, dass ein intelligenter Agent *autonom*, *perzeptiv* und *reaktiv* sein muss. Weitere Eigenschaften werden in den jeweiligen Arbeiten definiert und können zur Klassifikation verwendet werden. Neben den bereits genannten Attributen finden sich in der Literatur noch:

- *abgeschlossen*: Ein abgeschlossener Agent ist begrenzt im Umfang und man kann eindeutig ermitteln, ob ein bestimmter Teil des Systems zum Agenten gehört oder nicht. [MN05, MN06]

- *emotional*: Ein emotionaler Agent soll bei einem Beobachter die Illusion erzeugen, ein lebendiges Wesen zu sein. Nach [Bat94] ist der Schlüssel dazu, ein sichtbarer emotionaler Zustand innerhalb des Agenten. Diese Art von Agenten ist hauptsächlich interessant für direkt mit dem Menschen interagierende Systeme, z. B. einen Hausroboter.

- *mobil*: Ein mobiler Agent hat die Fähigkeit, sich innerhalb seiner Umwelt zu bewegen [Mil99]. Diese Umwelt kann ein Netzwerk von Computern sein, in dem er von einem Knoten zu einem anderen wechselt oder auch eine einfachere Zellstruktur, in dem der Agent die Zellen wechselt.

- *aufrichtig*: In [GK94] wird diese Eigenschaft so beschrieben, dass jeder Agent immer die Wahrheit sagen muss, wenn er mit anderen Agenten kommuniziert, wobei sich diese Wahrheit aus dem ergibt, was der Agent selbst als Wahrheit ansieht. Das wiederum hängt von seinem internem Zustand ab.

- *dynamisch*: Der Begriff fasst die Fähigkeiten zusammen, dass ein Agent sich klonen kann, sich mit einem anderen Agenten zu einem vereinen kann, geboren werden, oder sterben kann. Z. B. werden in [SSCJ98] die Bedingungen und die Vorteile von klonbaren Agenten in Systemen diskutiert, in denen Agenten durch zu viele Aufträge überlastet werden könnten.

Eine formale Definition und Klassifikation von konkreten Implementierungen von Agenten kann erst gegeben werden, wenn Einzelheiten der Modellierung des gesamten Systems bekannt sind, da die Struktur des Agenten und seine Eigenschaften unter anderem von der Umwelt und den Zielen abhängt. Informal lässt sich jedoch festhalten, dass in dieser Arbeit Agenten mindestens *autonom*, *perzeptiv* und *reaktiv* sein müssen. Die weiteren Eigenschaften sind optional. Abb. 2.1 zeigt das allgemeine Konzept eines Agenten mit den geforderten Eigenschaften. Mit einem *Sensor* können Informationen aus der Umwelt wahrgenommen werden. Diese *Wahrnehmungen* werden von der *Kontrollfunktion* verarbeitet und auf *Entscheidungen* abgebildet. Ein *Aktuator* setzt die Entscheidungen in *Aktionen* um, die wiederum eine *Wirkung* haben können, aber nicht

zwangsläufig haben müssen. Die Aktionen wirken einerseits auf den Zustand der Umwelt und/oder auf den *Datenzustand* innerhalb des Agenten, indem eine Abbildung aus altem Zustand und allen auf den Zustand wirkenden Aktionen auf einen neuen Zustand angewendet wird. Der Datenzustand ist ein optionaler Bestandteil des Agenten. Die Kontrollfunktion des Agenten kann optional auch einen *Kontrollzustand* besitzen.

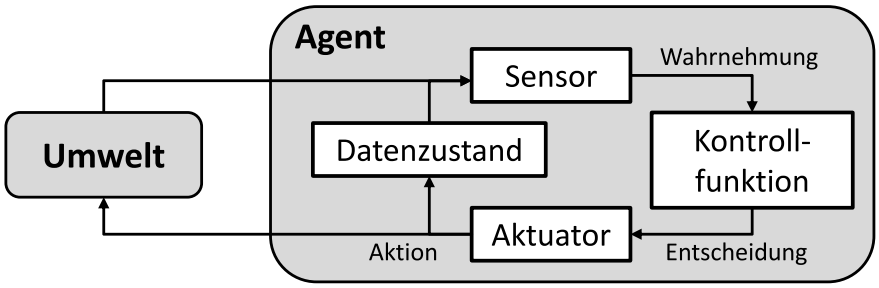

Abb. 2.1.: Schematische Darstellung eines einfachen autonomen, perzeptiven und reaktiven Agenten in Anlehnung an [RN04, S. 56] und [Woo02, S. 34].

Die Verarbeitung der Informationen aus der Umwelt und die Entscheidung und Umsetzung in Aktionen erfolgt ohne Eingriff einer anderen Einheit von außen. Die Wirkung auf die Zustände der Umwelt dagegen kann durch die Aktionen anderer Agenten beeinflusst werden.

Keine der bisher genannten Definitionen macht eine Einschränkung bei der Gestalt des Agenten. Es könnte sich um ein biologisches Wesen handeln, eine reale Maschine oder ein abstraktes Programm. Eine Kategorisierung in dieser Form nehmen Franklin und Graesser in [FG96] vor. Der autonome Agent als oberste Kategorie lässt sich in die Subkategorien biologischer Agent, Roboteragent und Berechnungsagent (computational agent) aufteilen. Es können jeweils weitere Subkategorien aufgestellt werden.

2.1.2 Klassifizierung von Agentenverhalten

Die gewünschten Eigenschaften des Agentenverhaltens variieren von MAS zu MAS. Einerseits könnte eine möglichst genaue Nachbildung des Verhaltens eines natürlichen Agenten gewünscht sein, andererseits könnte ein Verhalten gewünscht sein, das zur effizienten Lösung eines Problems führt. Deterministisches Verhalten konkurriert mit probabilistischem Verhalten, statisches Verhalten mit sich über die Zeit änderndem Verhalten, z. B. bei lernenden Agenten. Die Komplexität spielt eine Rolle für die Unterbringung des Verhaltens auf möglicherweise begrenzten Ressourcen und für die Entwicklung eines Verhaltens entweder per manuellem Design oder durch unterstützende automatische Verfahren.

Über unterschiedliche Eigenschaften des Verhaltens können Agententypen klassifiziert werden. Da das Verhalten ein Teil des gesamten Agenten ist, gibt es bei einer Klassifizierung des Verhaltens über verschiedene Typen zwangsläufig Überschneidungen mit der Klassifizierung der Agenten, wie in Abschn. 2.1.1 beschrieben. So unterscheidet Müller in [Mül96] beispielsweise zwischen *deliberativen*, *reaktiven* und *hybriden* Agenten. Letztere haben sowohl deliberative als auch reaktive Komponenten. Hier ist Reaktivität im Unterschied zu Abschn. 2.1.1 so zu verstehen, dass Agenten *ausschließlich* reaktiv sind und nicht deliberativ oder pro-aktiv.

Russell und Norvig [RN04, S. 70] bezeichnen eine konkrete Implementierung der Kontrollfunktion, also des Teils des Agenten, der vom Sensor Eingaben erhält und Ausgaben an den

Aktuator gibt, als *Agentenprogramm*. Es werden vier Typen von Agentenprogrammen unterschieden:

- *Einfache Reflex-Agenten* besitzen keinen internen Kontrollzustand. Sie haben einen festen Regelsatz, der den Zustand der Umwelt auf Aktionen abbildet. Das bedeutet, dieser Agent ist nicht deliberativ und nicht adaptiv.

- *Modellbasierte Reflex-Agenten* haben ein komplexeres Verhalten. Sie besitzen einen internen Kontrollzustand, das bedeutet, über diesen Zustand, kann er sich Wissen über die Umwelt merken. Er hat außerdem ein Wissen darüber, wie sich die Umwelt entwickelt und wie sich seine Aktionen auf den Zustand der Welt auswirken. Mit diesem Wissen kann er deliberativ handeln.

- *Zielbasierte Agenten* sind modellbasierte Agenten, die zusätzlich ein Ziel haben, das fest vorgegeben ist. Damit können Aktionen präferiert werden, die näher zum Ziel führen. Diese Agenten sind pro-aktiv.

- *Nutzenbasierte Agenten* besitzen statt Zielen eine Nutzenfunktion. Über diese Nutzenfunktion können alle Zustände einem Nutzen zugeordnet werden, nicht nur der Zielzustand. Dadurch wird laut Russell und Norvig ein rationales Verhalten erst möglich.

Schließlich führen die Autoren noch lernende Agenten ein. Ein lernender Agent kann einer der vier anderen Agententypen sein, mit der zusätzlichen Eigenschaft, sein Programm laufend adaptieren zu können. Dazu führen Sie zusätzliche Elemente ein, die die über die Sensoren wahrgenommene Situation bewerten und aufgrund dieses Feedbacks gegebenenfalls das ursprüngliche Programm ändern. Äußerst wichtig dabei ist auch ein Element, welches in der Lage ist, Aktionen zu generieren, die von den Aktionen nach Reflex-, zielbasierter oder nutzenbasierter Entscheidung abweichen.

Eine andere Einteilung in Typen macht Klügl in [Klü01, S. 21]. Sie beschreibt dort als Typen von Architekturen, was im Sinne von dieser Arbeit allerdings das Agentenprogramm ist. Unterteilt wird in vier Kategorien. *Deliberative* Architekturen entsprechen denen aus [Mül96], ebenso wie die *subkognitiven* Architekturen, die in [Mül96] *reaktiv* genannt werden. Dazu kommen *subsymbolische* Architekturen, die im Gegensatz zu den deliberativen Architekturen kein (symbolisches) internes Modell der Welt benutzen, um Pläne zu erstellen. Stattdessen ergibt sich das Wissen der Agenten über ihre Umwelt implizit aus den Mechanismen, die die Aktionen auswählen. Der vierte Typ sind die *kognitiven* Architekturen. Sie sind eigentlich ein Spezialfall der deliberativen Architekturen, deren Planungsweise auf *„kognitionswissenschaftlichen oder psychologischen Modellen“* beruht [Klü01, S. 27].

Zwischen *tropistischen* und *hysteretischen* Agenten unterscheiden Genesereth und Nilsson [GN89]. Sie klassifizieren damit die innere Struktur eines Agenten, d. h. diejenigen Elemente, die sein Verhalten steuern. Tropistische Agenten haben keinen internen Zustand und ihr Verhalten wird ausschließlich durch den momentanen Zustand ihrer Umgebung bestimmt. Es handelt sich also um einfache Reflex-Agenten wie oben beschrieben. Hysteretische Agenten sind solche mit einem internen Kontrollzustand und entsprechen den modellbasierten Reflex-Agenten aus [RN04].

2.1.3 Die Umwelt der Agenten

Neben der Klassifikation von Agenten, kann auch die Umwelt, in der sie sich befinden, in bestimmte Klassen eingeteilt werden. Russel und Norvig haben sich in [RN04, S. 66] dieser Aufgabe angenommen und wurden unter anderem in [Woo02, Klü01] (ältere Auflagen von [RN04]) zitiert. Danach gibt es fünf Kategorien nach denen man die Umwelt charakterisieren kann:

1. *Beobachtbarkeit*. Hier wird unterschieden, inwieweit der Agent den Zustand der kompletten Umwelt mit seiner Sensorik beobachten kann. Er kann *vollständige*, fehlerhafte, ungenaue, teilweise oder anders *eingeschränkte* Informationen haben.

2. *Determiniertheit*. Ob eine Umwelt *deterministisch* oder *stochastisch* ist, ist auch eine Frage der Perspektive. Wenn der aktuelle Zustand der Umwelt und die Aktionen der Agenten den Folgezustand eindeutig bestimmen, ist die Umwelt deterministisch, ansonsten stochastisch. Ist der aktuelle Zustand dem Agenten jedoch nicht vollständig bekannt, kann der Folgezustand auch in einer deterministischen Umwelt aus der Perspektive des Agenten nicht bestimmt werden.

3. *Historie*. Russel und Norvig nennen eine Umwelt *episodisch*, wenn der Folgezustand der Umwelt nur vom aktuellen Zustand und den Aktionen der Agenten abhängt. Sie ist *sequenziell*, wenn die Historie der Agentenaktionen, unabhängig vom aktuellen Zustand der Umwelt, eine Rolle spielt.

4. *Dynamik*. Als *dynamisch* wird eine Umwelt klassifiziert, wenn sich ihr Zustand ändert, während der Agent noch dabei ist, aufgrund des vorigen Zustands eine Aktion auszuwählen, sonst ist sie *statisch*.

5. *Stetigkeit*. In einer *diskreten* Umwelt, gibt es eine diskrete Menge von möglichen Zuständen (Wahrnehmungen). Gibt es Zustände, die stetige Werte annehmen können, so wird die Umwelt als *stetig* bezeichnet.

Eine sechste Kategorie zur Einteilung der Umwelt wird in [KFH04] genannt:

6. *Räumlichkeit*. Die Autoren unterscheiden zwischen *nicht-räumlichen* („*non-spatial*“), *räumlich diskreten* („*discrete spatial*“) und *räumlich kontinuierlichen* („*continuous spatial*“) Umwelten. Bei nicht-räumlichen Umwelten gibt es eine Topologie, die die Position eines Objekts festlegt, aber nicht auf einem Raum im mathematischen Sinne basiert. In einer räumlich diskreten Umwelt wird der Raum in diskrete Abschnitte eingeteilt, mit dessen Koordinaten die Positionen aller Objekte beschrieben werden können. In einer räumlich kontinuierlichen Umwelt kann die Position eines Objekts/Agenten mit Koordinaten aus rationalen Zahlen beschrieben werden (auch wenn das nur im Modell möglich ist, weil in einer konkreten Simulation auf digitalen Rechnern keine unendlichen (Zwischen)räume dargestellt werden können).

In Multiagentensystemen sind die Eigenschaften und das Design der Umwelt ein wichtiger Bestandteil, der über das reine Bereitstellen einer externen Infrastruktur, auf der sich die Agenten befinden, hinausgeht. In vielen MAS wird z. B. die Umwelt als Ressource für indirekte Kommunikation genutzt. In [WOO07] wird argumentiert:

„Yet, while the notion of environment is understood to some degree in the MAS community, it is not well defined. Researchers and engineers associate the environment with an amalgam of resources, services, infrastructure, and so on. For the most part, the environment is an implicit part of MAS that is often dealt with in an ad hoc way.“

Aber da die Umwelt für viele Bereiche des MAS zuständig sei, sei es sinnvoll sich über die Rolle der Umwelt Gedanken zu machen und beim Entwurf von MAS die Umwelt als expliziten Teil zu betrachten. Bandini et al. beschreiben in [BMV09] einige Zuständigkeiten der Umwelt, speziell auf die Simulation von MAS bezogen. So muss die Umwelt die physische Struktur des Gesamtsystems vergegenständlichen. Sie muss Zugriff auf Objekte des Systems ermöglichen, die nicht als Agent modelliert sind. Sie muss die Sensorik und die Aktionen der Agenten unterstützen. Und sie muss die Dynamik der Objekte, die sich unabhängig von den Agenten verändern können, verwalten.

Eine etwas andere Sichtweise findet sich in [KFH04]. Dort wird die Umwelt inklusive der Agenten als simulierte Umwelt („*simulated environment*“) bezeichnet. Um dieses Modell der Simulation zu einer lauffähigen Simulation zu machen benötigt man eine Simulationsumgebung („*simulation environment*“). Diese Unterscheidung soll beim Entwurf und der Implementierung von MAS helfen, unter anderem, weil dadurch modellunabhängige Simulationsumgebungen möglich sind.

Was in [KFH04] als simulierte Umwelt bezeichnet wird, wird in dieser Arbeit *Agentenwelt* (oder kurz: *Welt*) genannt. Die Agentenwelt beinhaltet alle Agenten und alles, was die Agenten potenziell wahrnehmen und/oder verändern könnten. Die *Umwelt* ist in dieser Arbeit aus Sicht eines Agenten definiert und beinhaltet die Agentenwelt mit allen Agenten exklusive dem Agenten selbst (Abb. 2.2). Der für den Agenten zu einem bestimmten Zeitpunkt wahrnehmbare Ausschnitt seiner Umwelt ist kleiner oder gleich der Umwelt und kann sich über den Zeitverlauf verändern. Gleiches gilt für den zu einem bestimmten Zeitpunkt für den Agenten modifizierbaren Ausschnitt der Umwelt, der nicht mit dem wahrnehmbaren Ausschnitt übereinstimmen muss. Wie bei den Agenten in Abschn. 2.1.1 kann eine vollständige Einordnung der Agentenwelt bzw. Umwelt in die jeweiligen Kategorien erst mit der Definition eines konkreten MAS erfolgen.

In realen Systemen, z. B. mit Robotern als Agenten, wird zwangsläufig durch die Digitalisierung und durch die beschränkte Sensorik der Raum diskretisiert und die Beobachtbarkeit der Welt beschränkt. An dieser Stelle lässt sich bereits sagen, dass aus diesem Grund in dieser Arbeit nur MAS mit räumlich diskreten, statischen Umwelten mit eingeschränkter Beobachtbarkeit behandelt werden. Dies ist mit dem gewählten Modell des Zellularen Automaten sehr gut vereinbar, der durch seine Zellstruktur einen diskreten Raum abbildet, durch die generationsweise Berechnung keine dynamischen Zustandsänderungen zulässt und durch seine lokale Nachbarschaft eine vollständige Beobachtung der Umwelt verhindert.

2.1.4 Eigenschaften von Multiagentensystemen

Ein MAS wird in [JSW98] definiert als ein lose verbundenes Netzwerk von Problemlösern (Agenten), die zusammenarbeiten, um Probleme zu lösen, die sie alleine aufgrund ihres Könnens und Wissens nicht lösen könnten. In dieser Arbeit wird ein MAS lediglich als ein (lose verbundenes) Netzwerk von Agenten betrachtet, die sich in einer Umwelt befinden und ein bestimmtes Verhalten haben. Die Einschränkung, dass die Agenten nicht alleine fähig sind, Probleme zu lösen, oder dass sie überhaupt Probleme lösen, wird hier nicht gemacht, da sonst einige Sys-

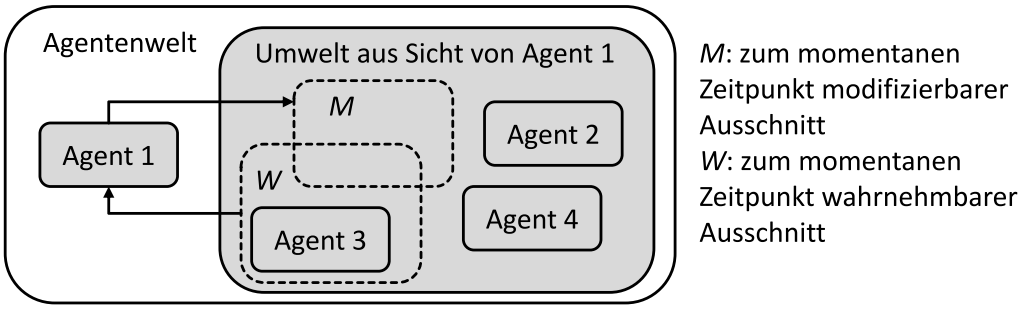

Abb. 2.2.: Die Agentenwelt und die Umwelt aus Sicht eines Agenten.

teme, deren primäre Funktion nicht in der Lösung eines Problems liegt (z. B. die Nachbildung eines natürlichen MAS zur Beobachtung und zum Verstehen von bestimmten Phänomenen in Gesellschaften), nicht unter die Kategorie MAS fallen würde, obwohl es aus miteinander agierenden Agenten besteht. Der Grundgedanke von kooperierenden Agenten zur Problemlösung ist dennoch auch in dieser Arbeit der zentrale Gedanke und führt zur Formulierung von Motiven einer Implementierung eines MAS oder speziell eines Systems zur kooperativen verteilten Problemlösung (Cooperative Distributed Problem Solving) [DLC89]:

- schnellere Lösungen von Problemen durch Parallelität
- eine größere Bandbreite an lösbaren Problemen durch geteilte Ressourcen (z. B. die Weitergabe von Informationen unter den Agenten oder die Nutzung von unterschiedlichen Fähigkeiten der Agenten)
- robustere Lösungsverfahren durch mehrfaches Ausführen von Aufgaben (evtl. mit unterschiedlichen Methoden)

Nicht jedes MAS ist in diesem Sinne *kooperativ*. Je nach Anwendung gibt es MAS mit *konkurrierenden* Agenten oder Agentengruppen, die jede(r) für sich ein eigenes Ziel hat. In [HS99] betrachten Huhns und Stevens MAS mit Fokus auf Kommunikations- und Interaktionsfähigkeiten der Agenten und unterscheiden kooperative und konkurrierende MAS unter dem Oberbegriff *Koordination*. In der Literatur finden sich aber auch Diskussionen, die ein konkurrierendes Verhalten ausschließen bzw. ein nicht-konkurrierendes annehmen [RG85]. Unter dem Begriff „*benevolent agent assumption*“ verbirgt sich dort, dass alle Agenten in einem System ein gemeinsames oder zumindest keine sich widersprechenden Ziele haben. Verhalten sich die Agenten kooperativ, können sie sich gegenseitig von Nutzen sein und das Gesamtverhalten im Sinne eines gemeinsamen Ziels verbessern oder sogar erst die Erreichung des Ziels möglich machen. Wenn sich solche positiven Effekte aus dem Zusammenwirken der einzelnen Agenten ergeben, spricht man teilweise auch von *Synergieeffekten* [BZN04].

Emergenz und Selbst-Organisation. Ein MAS als Ganzes ist dezentral kontrolliert, da ja die Agenten alle autonom sind. Dem würde eine zentrale Kontrolleinheit widersprechen. In einem dezentral kontrollierten System mit autonomen Einheiten, die in vielen Fällen eine beschränkte Sicht auf die Umwelt haben, entsteht häufig ein *Emergenz* genanntes Phänomenen, das im Folgenden beschrieben wird.

Das globale Verhalten des kompletten MAS ist eine logische Folge der lokalen Aktionen der einzelnen Agenten. Dennoch ist es oft nicht möglich das globale Verhalten auf das lokale Verhalten der Agenten zu reduzieren (Irreduzibilität), wenn man die Agenten einzeln betrachtet. Man

kann auch umgekehrt formulieren, dass aus den isoliert betrachteten lokalen Verhaltensregeln nicht direkt auf das globale Verhalten des Systems geschlossen werden kann. Für das letztlich entstehende globale Verhalten ist die Interaktion zwischen den Agenten von entscheidender Bedeutung. Auf die Schwierigkeit, die Funktionsweise eines dezentral kontrollierten Systems komplett zu ergründen, weist Gordon in [Gor07] hin.

Nach [DH04, DH05a] ist *Emergenz* das dynamische Entstehen eines globalen Verhaltens durch Interaktionen der lokalen Bestandteile (Agenten), wenn das globale Verhalten bezogen auf die lokalen Bestandteile *neuartig* ist. Schon Aristoteles stellte fest [Ari07, S. 217]:

*„Dasjenige, was so zusammengesetzt (*sỳnholon*) ist, dass das Ganze eins ist, nicht wie ein Haufen, sondern wie die Silbe, ist noch etwas anderes außer den Elementen. Die Silbe nämlich ist nicht einerlei mit ihren Elementen (*stoicheîa*), das* ba *nicht einerlei mit* b *und* a*, ebensowenig Fleisch mit Feuer und Erde;“*

Das Prinzip, dass das Ganze mehr ist als die Summe seiner Teile, lässt sich in der Natur häufig beobachten. Zum Beispiel ist die Temperatur des Wassers eine emergente Eigenschaft von dessen Molekülen. Ein einzelnes Molekül besitzt keine messbare Temperatur, ein ganzes Glas Wasser schon, obwohl es nur aus Molekülen besteht. Ameisen finden kürzeste Wege zu Futterquellen nur mittels kollektivem Verhalten, isoliert man eine einzelne, ist diese Fähigkeit nicht mehr aus ihrem individuellen Verhalten ersichtlich [GADP89]. Emergenz ist auch mit komplexeren Agenten möglich, wie musizierende Menschen in einem Orchester, Sportler in einer Mannschaft, etc. Auch in der Technik begegnet man der Emergenz tagtäglich. So ist ein aus Millionen von Pixeln aufgebautes Bild auf einem Bildschirm die emergente Eigenschaft des Gesamtsystems bestehend aus den einzelnen Pixeln. Emergenz beruht, wie die Synergie auch, auf den Wirkungen des Zusammenspiels der einzelnen Agenten. In Abgrenzung zur Synergie wird jedoch nicht unbedingt ein positiver Effekt für ein Gesamtziel gefordert. Dafür sind für Synergieeffekte weder eine Neuartigkeit, noch Irreduzibilität notwendig. Eine ausführlichere Behandlung des Themas Emergenz findet sich in [Hol00].

Selbst-Organisation ist ein Begriff, den man oft im Zusammenhang mit Emergenz liest. Gemeint ist damit, dass ein System ohne äußere Kontrolle oder Beschränkungen eine stabile Struktur erreicht, d. h. dass die Struktur aus den systeminternen Beschränkungen und Mechanismen folgt [DFH$^+$04]. In [DH04, DH05a, DGK06] werden Emergenz und Selbst-Organisation zueinander in Beziehung gesetzt und die Unterschiedlichkeit beider Konzepte betont. Beide Phänomene sind demnach dynamische Prozesse über die Zeit. Den Hauptunterschied sehen die Autoren darin, dass es bei der Emergenz einen lokal-zu-global Effekt gibt, der eine neuartige globale Eigenschaft hervorruft, sowie die Irreduzibilität dieser globalen Eigenschaft und die dezentrale Kontrolle. Diese Eigenschaften sind für die Selbst-Organisation nicht notwendig. Im Gegensatz dazu sind die wesentlichen Eigenschaften eines selbst-organisierenden Systems das Erreichen einer höheren Ordnung, das Fehlen der externen Kontrolle und die Anpassungsfähigkeit an Änderungen. In der Literatur wird teilweise zwischen *stark* und *schwach selbstorganisierenden* Systemen unterschieden. In Systemen mit starker Selbst-Organisation kann es weder eine interne noch eine externe zentrale Kontrolle geben, in schwachen Systemen kann eine interne zentrale Kontrolle existieren [DGK05]. Eine Reihe von technischen Applikationen für selbstorganisierende MAS werden in [BCHM06] vorgestellt.

In MAS treffen Selbst-Organisation und Emergenz oft zusammen, z. B. in Systemen mit *Schwarmintelligenz* (auch *kollektive Intelligenz*). Während Emergenz als Oberbegriff auch Systeme mit nicht-autonomen Einzelteilen umfasst, werden zur Schwarmintelligenz meist nur Sys-

teme mit autonomen Individuen gezählt. Klassische Beispiele in der Natur sind Schwärme von Vögeln, Fischen oder Insekten, die in ihrer Form als Schwarm von außen betrachtet eine intelligente Supereinheit bilden. Reynolds konnte z. B. in [Rey87] ein Formationsverhalten, wie bei Vogelschwärmen beobachtbar, mit relativ simplen Regeln für die Individuen nachbilden und simulieren. Eine ausführlichere Diskussion findet sich in [KES01] oder [BDT99]. Die Betrachtung von biologischen Schwärmen, auf dessen Prinzipien die künstlichen Schwärme beruhen, und eine Kategorisierung von kollektivem Verhalten liefern Garnier et al. in [GGT07]. Sie stellen weitere Phänomene der Schwarmintelligenz vor: *Koordination* ist die Organisation der Agenten in Raum und Zeit, um z. B. eine örtliche Verlegung der Position des ganzen Schwarms zu erreichen (anders als bei [HS99]). *Deliberation* ist die Fähigkeit, eine kollektive Entscheidung zwischen mehreren Optionen zu treffen. *Kollaboration* tritt auf, wenn es Agenten mit unterschiedlichen Fähigkeiten/Aufgaben gibt, die durch Zusammenarbeit zum globalen Ziel führen.

Eine bestimmte Form der Selbst-Organisation in einem dezentral kontrollierten MAS ist die *Stigmergie*. Damit bezeichnet man eine indirekte Form der Kommunikation zwischen den Agenten, um sich durch gegenseitige Beeinflussung zu koordinieren. Dabei modifizieren die Agenten ihre lokale Umgebung so, dass die Modifikation für andere Agenten sichtbar wird und diese zu bestimmten Aktionen veranlasst. So kann auch eine sich selbst verstärkende Wirkung auftreten. Natürliche Beispiele, die als Inspiration für künstliche Systeme dienen, sind unter anderem nahrungssuchende Ameisen, die Pheromone auf dem Weg hinterlassen [VVKB01].

Selbst-X-Eigenschaften. Ein von IBM (International Business Machines Corporation) geprägtes Paradigma von Rechensystemen, die sich selbst verwalten können, ist das *Autonomic Computing* Paradigma [KC03]. Darin werden Eigenschaften von autonomen Systemen beschrieben, die auch in Bezug auf MAS, welche in einigen Fällen als autonome Systeme klassifiziert werden können (z. B. selbstorganisierende MAS), auftreten können. Genannt werden vier Prinzipien, die ein System als autonom qualifizieren:

- *Selbst-Konfiguration*. Automatisierte Konfiguration aller Komponenten des Systems aufgrund von spezifizierten globalen Zielen.
- *Selbst-Optimierung*. Kontinuierliche Versuche aller Komponenten, ihre Leistung zu verbessern.
- *Selbst-Heilung*. Probleme werden automatisch erkannt und repariert.
- *Selbst-Schutz*. Automatische Abwehr von äußeren Attacken und inneren Fehlern durch Antizipation.

In [HRT+05] werden diese Prinzipien aufgegriffen, um einen autonomen künstlichen Schwarm, bestehend aus mehreren kleinen Raumschiffen, zu beschreiben, der in künftigen NASA Missionen zum Einsatz kommen soll.

Ein weiteres Forschungsgebiet, das sich mit komplexen, autonomen Systemen und Selbst-X-Eigenschaften beschäftigt ist *Organic Computing* [Wür08, OC003]. In dem Forschungszweig geht es um die Entwicklung von robusten, künstlichen Systemen, die Emergenz und Selbst-X-Eigenschaften besitzen. Hierbei ist auch festzuhalten, dass Emergenz oder Selbst-Organisation per se nichts Gutes oder Schlechtes ist, das man anstreben oder vermeiden sollte. Außer zielführenden Effekten, wie sie sich in der Natur durchgesetzt haben, kann es auch unerwünschte emergente Effekte geben, wie z. B. die Anhäufung von Hühnern um ein verwundetes Huhn, um es totzupicken [RPS08]. Entscheidend ist nun der Wunsch, dass nur gewünschte emergente Eigenschaften zutage treten, also eine Art von kontrollierter Emergenz notwendig ist [Sch05]. Für

eine kontrollierte Emergenz muss diese formal definiert und gemessen werden können. Ansätze für eine solche Formalisierung und automatischen (nicht menschlichen) Messung von Emergenz über Entropie und Ordnung beschreiben Müller-Schloer et al. in [MM06, MS08].

Ein Teilziel dieser Arbeit ist es, heuristische Methoden zu entwickeln, um Lösungen (im Sinne von Agentenverhalten) für emergente und selbst-organisierende MAS zu erhalten. Dabei sollen auch die oben beschriebenen Formen der Selbst-Organisation wie Stigmergie, Koordination etc. ermöglicht werden. Im Sinne des Organic Computing gilt es, die unerwünschten emergenten Eigenschaften im Zuge der Entwicklung der heuristischen Methoden zu vermeiden.

2.1.5 Anwendungen von Multiagentensystemen

MAS finden in zahlreichen Disziplinen der Wissenschaft Anwendung. So sind MAS in der Ökonomie (Spieltheorie) [PW02], Ökologie [BL04], Biologie [AS07] und Politik- und Sozialwissenschaft [YHM+07] zu finden, wo sie eingesetzt werden, um komplexe Zusammenhänge zu verstehen und gegebenenfalls Optimierungsansätze zu finden. Im technischen Bereich gibt es MAS beispielsweise in der Robotik oder der Softwaretechnik. Insgesamt findet man MAS in mehr Bereichen als man an dieser Stelle aufzählen könnte. Die hier genannten Beispiele sollen verdeutlichen, dass es sich bei MAS um ein Konzept mit einer außergewöhnlich großen Anwendungsvielfalt handelt.

In Kap. 1 wurden zwei grundsätzlich verschiedene Anwendungsformen von MAS unterschieden. Aufgrund der Fülle von verschiedenen Anwendungen, soll diese Unterscheidung zwischen einerseits Nachbildungen natürlicher Systeme und andererseits künstlichen Systemen nun genauer beschrieben und weiter unterteilt werden, um eine genaue Einordnung der MAS aus dieser Arbeit vornehmen zu können. Ebensogut könnte man auch nach anderen Kriterien, wie z. B. Softwareagenten und Hardwareagenten oder biologischen und mechanischen MAS, unterscheiden. Da hier aber eher der Zweck des MAS im Vordergrund stehen soll, und nicht dessen physische Beschaffenheit, wird die erste hier aufgeführte Variante gewählt.

Nach [KFH04] können MAS entweder einen *erklärenden* oder einen *vorhersagenden* Zweck haben. Das bezieht sich auf MAS, die natürlich vorkommende Systeme nachbilden. Im ersten Fall ist das globale Verhalten im Prinzip bekannt, aber noch keine Erklärung für dessen Entstehen gefunden. Der Entwickler eines solchen Systems steht also vor der Aufgabe, das lokale Verhalten der Agenten nach einer Theorie so zu modellieren, dass das gewünschte globale Verhalten entsteht. Durch Simulation kann die Theorie dann bestätigt werden. Im zweiten Fall ist das lokale Verhalten der Agenten bekannt und es soll per Simulation eine Vorhersage für globales Verhalten gemacht werden (Tab. 2.2). Beispiele für erklärende MAS sind Verkehrssimulationen wie in [NS92, SS93], wo die Entstehung von Staus erklärt werden soll. Beispiele für MAS zur Vorhersage sind [Her07], wo Herrler Abläufe in Krankenhäusern simuliert, und [LD09]. In letzterem werden Verkehrssimulationen für Kreuzungen untersucht, um Reisezeiten vorherzusagen. In vielen Fällen, wie z. B. der Verkehrssimulation, ist es so, dass wenn einmal eine Theorie durch ein erklärendes MAS bestätigt ist, anschließend diese Systeme benutzt werden, um Vorhersagen zu treffen und letztendlich Prozesse zu optimieren, entweder durch Optimierung des Agentenverhaltens (z. B. eine Veränderung der Maximalgeschwindigkeit) oder durch Veränderung der äußeren Bedingungen (z. B. Hinzufügen einer weiteren Fahrspur).

Eine andere Einsatzmöglichkeit von MAS sind künstliche Systeme, die eine bestimmte Aufgabe erfüllen sollen. Die Erfüllung der Aufgabe ist das globale Ziel. Gesucht sind nun *problem-*

Tab. 2.2.: Anwendungsformen von MAS.

Form der Anwendung	lokales Verhalten	Interaktionsmechanismen	globales Verhalten
erklärend	Theorie soll per Simulation bestätigt werden	soll herausgefunden werden	ist aus der Natur bekannt
vorhersagend	ist bekannt	ergibt sich aus lokalem Verhalten	wird beobachtet
problemlösend	wird gesucht (zielführend modelliert)	ergibt sich aus lokalem Verhalten	ist als Ziel spezifiziert
erzeugend	wird beliebig vorgegeben	wird beobachtet	wird beobachtet

lösende Agenten, deren lokales Verhalten das gewünschte globale Verhalten induziert. Anders als bei den erklärenden MAS gilt es hier nicht, eine Theorie zu überprüfen, sondern ein globales Ziel zu erreichen, d. h. es soll nicht erklärt werden, warum ein globales Verhalten entsteht, sondern herausgefunden werden, wie man es erzeugen und ggf. auf optimale Weise erzeugen kann (Tab. 2.2). Beispielsweise suchen Di Caro und Dorigo in [DD97] nach Routingalgorithmen, Fey und Schmidt implementieren MAS, um Bilder zu verarbeiten [FS05] und die bereits in Abschn. 2.1.4 erwähnten NASA-Schwarmmissionen ([HRT+05, THRR04]) sind ebenfalls ein Beispiel für problemlösende MAS. Neben diesen anwendungsorientierten Problemen, gibt es auch „hausgemachte“ Probleme, also solche, die nur in MAS auftreten. So beschreiben Lin et al. in [LMA05] das *Rendezvous Problem*, bei dem sich die mobile, autonome Agenten ohne explizite Kommunikationsfähigkeiten in einem räumlichen System an einem unbestimmten Ort treffen sollen.

Künstliche Systeme mit einem vorgegebenen lokalen Agentenverhalten, die nicht über ein globales Ziel verfügen, werden verwendet, um ein grundsätzliches Verständnis von der Interaktion zwischen Agenten und der Entstehung emergenter Phänomene zu gewinnen. Da das Erzeugen der globalen Phänomene hier das Entscheidende ist, kann man solche Systeme als *erzeugend* bezeichnen. Zum Beispiel wird in [IN97] das Verhalten von Räuber-Beute-Systemen analysiert, die nicht einem natürlichen System nachempfunden sind. Stattdessen wird erst im Nachhinein das globale Verhalten mit dem in der Natur vorkommenden Systemen verglichen.

In dieser Arbeit geht es ausschließlich um problemlösende MAS und die Entwicklung von Methoden zur Suche nach geeignetem Agentenverhalten.

2.2 Grundlagen von Zellularen Automaten

Der Zellulare Automat ist ein Berechnungsmodell, dessen Grundlagen in den 50er Jahren von Stanislaw Ulam und John von Neumann gelegt wurden [BSW85]. Von Neumanns Grundlagenarbeit zu CA als universellem Berechnungsmodell wurde 1966 posthum veröffentlicht [Neu66]. Bei von Neumann's CA handelt es sich um ein unendliches, zweidimensionales Gitter von Zellen (quadratischer Form), die mit ganzzahligen Koordinaten versehen sind und 29 mögliche Zustände besitzen. Zudem gibt es eine *Nachbarschaft*, die die Zelle selbst und die direkt angrenzenden vier Zellen in jeder Richtung einschließt (Abb. 2.3(a)). Jede Zelle besitzt den gleichen endlichen Automaten mit einer Zustandsübergangsfunktion, die den Zustand zum Zeitpunkt $t+1$ aus den Zuständen der Nachbarschaft zum Zeitpunkt t bestimmt. Der CA ist also ein räumlich und zeit-

lich diskretes, dynamisches System. Ein CA ist außerdem deterministisch und sein Verlauf über die Zeit abhängig von den initialen Zuständen der Zellen. Eine formale Definition von CA, die die Nachbarschaftsstruktur, deren Interpretation als geometrische Form der Zellen und die Anzahl der Zustände offen lässt, aber die räumlich und zeitlich diskrete Dynamik beibehält, ist in [Vol79, S. 15ff.] gegeben. Ganz ähnlich wird hier definiert:

Definition 2.1 *Ein Zellularer Automat A ist ein Quadrupel* $A = (d, N, S, \phi)$. *d ist die* Dimension *eines kartesischen Koordinatensystems (mit ganzzahligen Koordinaten), in dem jede Zelle des CA eine eindeutige Koordinate zugeordnet hat. Die komplette Gitterstruktur, die alle Zellen umfasst, wird auch* Zellulares Feld *genannt. Die* Nachbarschaft *N ist eine endliche Menge von d-Tupeln, die die relativen Positionen der Nachbarzellen einer Zelle angeben. Das* Zustandsalphabet *S ist eine endliche Menge von Zuständen, die eine Zelle annehmen kann.* ϕ *ist die* Transitionsfunktion $S^{|N|} \rightarrow S$, *die den Folgezustand (zum Zeitpunkt t+1) einer Zelle aus den momentanen (zum Zeitpunkt t) Zuständen der Nachbarzellen festlegt. Auf die Funktion wird auch mit* Zellregel *oder* Regel *referenziert.*

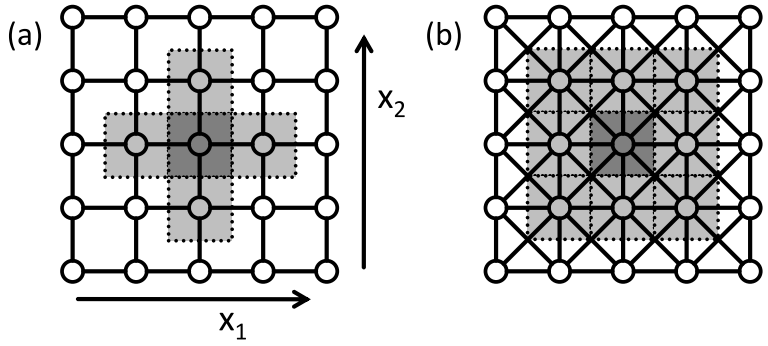

Abb. 2.3.: Bekannte Nachbarschaften im Zellularen Automaten: (a) zweidimensionale von Neumann-Nachbarschaft mit Radius 1, (b) zweidimensionale Moore-Nachbarschaft mit Radius 1. Die Nachbarschaft ist bezogen auf die mittlere Zelle.

Diese Definition des CA beschreibt sowohl die Topologie und die Nachbarschaft zwischen den Zellen mit den Variablen d und N, als auch die innere Struktur einer einzelnen Zelle mit den Variablen S und ϕ. Es ist nicht zwingend erforderlich, dass eine Zelle innerhalb ihrer eigenen Nachbarschaft liegt, dennoch liegt es in den meisten Anwendungen nahe, den Folgezustand vom bisherigen Zustand abhängig zu machen. In der in Abb. 2.4 gezeigten Struktur einer Zelle wird daher die Annahme gemacht, dass die Zelle selbst in der Nachbarschaft enthalten ist, so dass der eigene Zustand und die Zustände der weiteren $|N| - 1$ Nachbarn Eingangsparameter der Transitionsfunktion ϕ sind.

Vollmar fordert zudem, dass der CA einen unendlichen Raum in Zellen aufteilt und dass es einen Ruhezustand gibt, der durch die Transitionsfunktion wieder in den Ruhezustand führt. Der Ruhezustand kann dafür genutzt werden, einen imaginären „Rand“ zu erzeugen (nicht unbedingt im geometrischen Sinne), so dass bei der Berechnung der Folgezustände nur eine endliche Zahl von Zellen berücksichtigt werden muss. Da wir aber bei praktischen Anwendungen wie MAS keinen unendlichen Raum benötigen, werden diese Einschränkungen in dieser Arbeit nicht gemacht, was allerdings zur Folge hat, dass ein tatsächlicher „Rand“ entsteht, der eine besondere Behandlung erfordert. Im folgenden Abschnitt werden einige bekannte Nachbarschaftsmodelle beschrieben und danach Varianten des CA vorgestellt, die von der obigen

Definition abweichen und dadurch eine Randbehandlung und andere erweiterte Möglichkeiten zur Modellierung von Systemen schaffen.

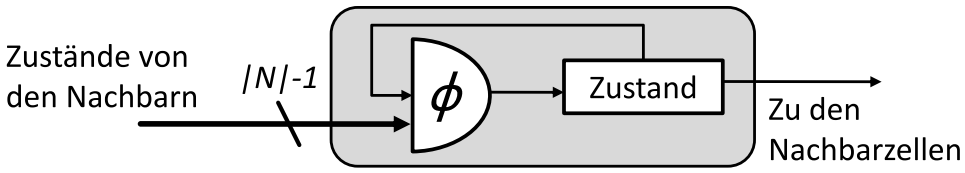

Abb. 2.4.: Struktur einer Zelle im uniformen Zellularen Automaten. Die Topologie des Zellularen Feldes (Dimension und Nachbarschaft) ist in diesem Schema nicht gezeigt.

2.2.1 Nachbarschaftsmodelle und Varianten des Zellularen Automaten

Die von Neumann-Nachbarschaft N_{vN} wird auch mit einem *Radius* r beschreiben, der den maximalen Abstand einer Zelle $(x_1, \ldots, x_d)$ zu den Nachbarn beschreibt. Formal ist die von Neumann-Nachbarschaft die Menge

$$N_{vN} = \{(y_1, \ldots, y_d) : \sum_{i=1}^{d} \left|y_i - x_i\right| \leq r\}.$$

Eine andere bekannte Nachbarschaft ist die Moore-Nachbarschaft N_M, in der auch die diagonal angrenzenden Zellen eines orthogonalen Gitters Nachbarn sind (Abb. 2.3(b)).

$$N_M = \{(y_1, \ldots, y_d) : \max(\left|y_1 - x_1\right|, \ldots, \left|y_d - x_d\right|) \leq r\}$$

Die Topologie der Zellen lässt sich durch einen Graphen mit Knoten und Kanten darstellen (wie in Abb. 2.3). Die Nachbarschaftsbeziehung wird dabei durch eine Kantenverbindung zweier Knoten symbolisiert. Netztopologien sind *regulär*, wenn alle Knoten des Graphs die gleiche Anzahl an Kanten besitzen. Sie sind *homogen*, wenn es einen Automorphismus gibt, der keinen Knoten auf sich selbst abbildet. Umgangssprachlich formuliert heißt das, dass der Graph von jedem Knoten aus betrachtet gleich aussehen muss bzw. dass es eine Darstellung der Topologie gibt, die durch ein immer wiederkehrendes Grundmuster gebildet werden kann.

Von Quadraten abweichende Interpretationen der räumlichen Gestalt der Zellen sind je nach Struktur der Nachbarschaft möglich, z. B. als zweidimensionale hexagonale und trianguläre Gitter. Dazu müssen entsprechend *reguläre*, *homogene* Netzstrukturen mit sechs oder drei Nachbarn pro Knoten verwendet werden (Abb. 2.5). Diese können auch auf orthogonale Koordinaten abgebildet werden. Eine formale Beschreibung der Nachbarschaft eines hexagonalen Gitters für $d = 2$ mit Radius r ist

$$N_{hex}^{d=2} = \{(x_1 + a, x_2 + b) : -r \leq a, b \leq r \wedge |a - b| \leq r\}.$$

Die Abbildung eines triangulären Gitters auf zweidimensionale kartesische Koordinaten lässt eine Nachbarschaftsbeschreibung nur mit Fallunterscheidung zu, da je nach Parität der Koordinaten einer Zelle die direkten Nachbarn relativ zur eigenen Position unterschiedlich liegen

(Abb. 2.5(b)). Zur formalen Beschreibung der Nachbarschaftsmenge werden der Übersicht halber zunächst einige Hilfsmengen eingeführt:

$$N_0 = \{(x_1 + a, x_2 + b) : -r \leq a + |b| \leq r \wedge |b| < \frac{r}{2}\},$$

$$N_{\pi(r)=1} = \{(x_1+a, x_2-\pi(x_1)\cdot\frac{r}{2}) : -\frac{r}{2} \leq a \leq \frac{r}{2}\} \cup \{(x_1+a, x_2+\pi(x_1)\cdot\frac{r}{2}) : -\frac{r}{2} \leq a \leq \frac{r}{2} \wedge \pi(a) = 1\},$$

$$N_{\pi(r)=-1} = \{(x_1 + a, x_2 + \pi(x_1) \cdot \left\lceil \frac{r}{2} \right\rceil) : -\frac{r}{2} < a < \frac{r}{2} \wedge \pi(a) = -1\},$$

wobei die Paritätsfunktion $\pi(i) = 1$, falls i gerade ist, sonst $\pi(i) = -1$. Die Nachbarschaftsmenge des zweidimensionalen triangulären Gitters wird nun wie folgt definiert:

$$N_{tri}^{d=2} = \begin{cases} N_0 \cup N_{\pi(r)=1} & \text{, falls } \pi(r) = 1 \\ N_0 \cup N_{\pi(r)=-1} & \text{, falls } \pi(r) = -1. \end{cases}$$

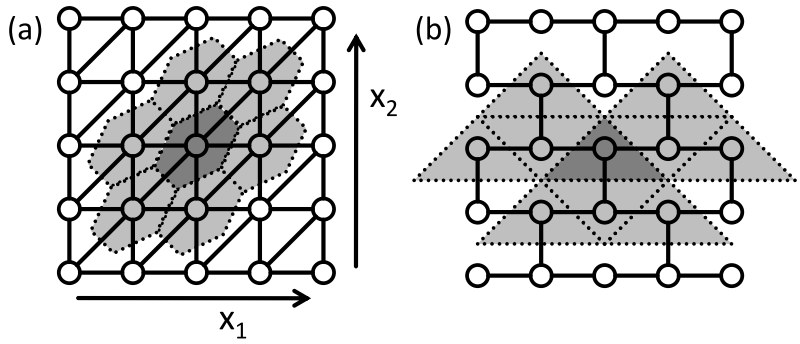

Abb. 2.5.: Zweidimensionale hexagonale und trianguläre Nachbarschaft im Zellularen Automaten. Die Nachbarschaft ist bezogen auf die mittlere Zelle. In der Gitterstruktur, die die kartesischen Koordinaten repräsentiert, ist eine von Quadraten abweichende geometrische Interpretation der Zellen abgebildet, (a) die hexagonale Nachbarschaft mit Radius 1 beinhaltet 7 Zellen. (b) die trianguläre Nachbarschaft mit Radius 2 beinhaltet 10 Zellen.

Die Nachbarschaft eines CA ist statisch und lokal in dem Sinne, dass nicht alle Zellen des CA in der Nachbarschaft einer Zelle sind und dass meistens nur die in der geometrischen Interpretation lokal angrenzenden Zellen dazu gehören, auch wenn das Modell theoretisch CA zulässt, die diese Einschränkungen nicht haben. Eine Erweiterung des Modells, die eine dynamische und globale Nachbarschaft *explizit* zulässt, ist der *Globale Zellulare Automat* (GCA) [HVW00, Hee07], in welchem die Nachbarschaft in Abhängigkeit des Zellzustandes oder anderer veränderlicher Parameter (z. B. die Zeit t) definiert wird. Dadurch, dass die Nachbarschaft für alle Zellen gleich definiert ist, können nur reguläre, homogene Netzstrukturen, also triangulär, orthogonal und hexagonal, für CA verwendet werden. Eine Diskussion über CA mit irregulären oder regulären inhomogenen Gitterstrukturen, die inhomogene Nachbarschaften für jede Zelle erlauben, findet sich unter anderem in [OT00], [OM07] oder [Hoc98, S. 9f.].

Bei endlich großen Feldern stellt sich zusätzlich die Frage, wie ein reguläres Netz am Rand aussieht. Prinzipiell gibt es hier zwei Möglichkeiten. Eine Möglichkeit ist das zyklische Schließen des Feldes, so dass es gar keine Zellen am Rand mehr gibt. Dieses Prinzip wird *Wrap-around* (engl. für Umgriff) genannt. Aus einem zweidimensionalen endlichen Raum wird dann ein Torus. Sei l_i die Länge des Zellularen Feldes in der i-ten Dimension, dann bedeutet ein Wrap-around für die relative Positionsangabe in kartesischen Koordinaten, dass eine Veränderung um a in der i-ten Dimension mit $(x_i + a \mod l_i)$ berechnet wird. Die andere Möglichkeit ist die Verwendung einer speziellen Regel für die Zellen am Rand, die notgedrungen eine andere Nachbarschaft haben als die Zellen in der Mitte des Zellularen Feldes.

Die Transitionsfunktion des CA ist zwar in Definition 2.1 für alle Zellen gleich definiert, jedoch wird diese Einschränkung in einigen Arbeiten zu CA aufgehoben [Sip94, Sip97, ST99]. CA mit einheitlichen Regeln für alle Zellen werden *uniform* genannt. CA, die unterschiedliche Regeln für Zellen haben, werden *nicht-uniform* genannt[1]. Die Begründung für die Verwendung von nicht-uniformen CA ist, dass sowieso in jeder Zelle eine Kopie des Automaten, der die Transitionsfunktion realisiert, vorhanden sein muss. Daher sind keine zusätzlichen Ressourcen nötig, aber das Modell um viele Möglichkeiten, wie z. B. *ortsabhängige* oder *zeitabhängige* Regeln, erweitert. Für Randzellen in Systemen ohne Wrap-around muss es also eine ortsabhängige nicht-uniforme Regel geben. Formal heißt das, dass es eine Menge von Paaren $R = \{(N_0, \phi_0), (N_1, \phi_1), \ldots, (N_n, \phi_n)\}$ geben muss, die jeweils für eine Teilmenge des Zellularen Feldes eine Nachbarschaft N_i und eine zugehörige Zellregel $\phi_i : S^{|N_i|} \to S$ definieren. Die Dimension d und das Zustandsalphabet S bleiben weiterhin für alle Zellen gleich, so dass der Zellulare Automat definiert wird durch das Quadrupel $A_{\text{nu}} = (d, S, R, \delta)$, wobei die Funktion $\delta : \mathbb{Z}^d \to R$ jeder Zelle (identifiziert über ihre Koordinaten) ein Paar aus Nachbarschaft und Transitionsfunktion zuordnet. Zu beachten ist, dass nur solche Zuordnungen erlaubt sind, die keine nichtexistenten Koordinaten als Nachbarpositionen zur Folge haben.

Eine andere Möglichkeit ortsabhängige Regeln einzuführen ist es, die Koordinaten der Zelle direkt als Parameter der Transitionsfunktion zu verwenden: $\phi : S^{|N|} \times \mathbb{Z}^d \to S$, ortsabhängige Nachbarschaften würde das allerdings nicht mit einschließen. Zeitabhängige Regeln erfordern, dass der globale (für alle Zellen gleiche) Parameter t in der Transitionsfunktion berücksichtigt wird. Man kann neben Zeitparametern auch andere globale Parameter (z. B. Anzahl der Zellen o. ä.) verwenden. Seien alle globale Parameter gegeben durch $\gamma_0, \gamma_1, \ldots, \gamma_m$ und $\Gamma_0, \Gamma_1, \ldots, \Gamma_m$ die dazugehörigen Alphabete (also Mengen von möglichen Werten der Parameter), dann sind die Transitionsfunktion wie folgt: $\phi_i : S^{|N_i|} \times \Gamma_0 \times \Gamma_1 \ldots \Gamma_m \to S$. Alternativ können diese Parameter auch als Teil des Zellzustands lokal in jeder Zelle gespeichert sein. Dann braucht man keine globalen Parameter mehr, muss aber durch die lokalen Zellregeln sicherstellen, dass die Werte in allen Zellen konsistent bleiben. So bräuchte man für den Parameter t einen Zeitzähler in jeder Zelle (als Teil der Transitionsfunktion). Freilich können ortsabhängige und von globalen Parametern abhängige Regeln miteinander in einem CA kombiniert werden.

Die Regeln des CA sind deterministisch nach Definition 2.1. Es werden aber auch CA untersucht, deren Transitionsfunktion teilweise zufällig ist. In *Probabilistischen Zellularen Automaten* (PCA) [DK84][MS91][Wol86b, Kap. 4] sind die Zustandsübergänge mit einer bestimmten Wahrscheinlichkeit verknüpft. Dabei sind mehrere deterministische Regeln gegeben und der Folgezustand einer Zelle wird zufällig durch eine von diesen Regeln bestimmt, wobei die Regeln

[1] Jeder nicht-uniforme CA kann durch Erweiterung der Zustände und der Transitionsfunktion in einen uniformen CA umgewandelt werden.

nicht unbedingt alle die gleiche Auswahlwahrscheinlichkeit haben müssen. Formal beschrieben bedeutet das, für einen PCA müssen (ausgehend von Definition 2.1) anstelle der Funktion ϕ mehrere Transitionsfunktionen $\phi_0, \phi_1, \ldots, \phi_k$ und eine Wahrscheinlichkeitsfunktion $p(\phi_i)$ mit $\sum_{i=0}^{k} p(\phi_i) = 1$ angegeben werden. Zusätzlich können diese Regeln in nicht-uniformen CA orts- und/oder zeitabhängig gemacht werden, woraus eine verallgemeinerte, den CA aus Definition 2.1 enthaltende Definition eines nicht-uniformen, probabilistischen CA folgt:

Definition 2.2 *Ein nicht-uniformer Probabilistischer Zellularer Automat ist ein Quadrupel $A_{nu,p} = (d, S, R_p, \delta)$, mit der Dimension d und dem Zustandsalphabet S wie in Definition 2.1. R_p ist eine Menge von probabilistischen, nicht-uniformen Regelmengen $R_p = \{(N_0, \Phi_0, p_0), (N_1, \Phi_1, p_1), \ldots, (N_n, \Phi_n, p_n)\}$, wobei $\Phi_i = \{\phi_{i,0}, \phi_{i,1}, \ldots, \phi_{i,k}\}$ eine Menge von Transitionsfunktionen $\phi_{i,j} : S^{|N_i|} \times \Gamma_0 \times \Gamma_1 \ldots \Gamma_m \rightarrow S$ ist. Γ_i ist dabei das Alphabet des globalen Parameters γ_i. Für die Wahrscheinlichkeitsfunktion p_i gilt $\sum_{j=0}^{k} p_i(\phi_{i,j}) = 1$. Die Funktion $\delta : \mathbb{Z}^d \rightarrow R_p$ ordnet die probabilistischen, von globalen Parametern abhängigen Regelmengen den Zellen zu.*

Angenommen, eine Zelle mit den Koordinaten K sei über die Funktion $\delta(K) = (N_i, \Phi_i, p_i)$ einer Regelmenge zugeordnet. Die Struktur dieser Zelle ist in Abb. 2.6 gezeigt. Wie in Abb. 2.4 wird auch hier angenommen, dass die Zelle selbst in ihrer Nachbarschaft N_i liegt. Alle Transitionsfunktionen $\phi_{i,j}$ haben dieselben Eingangsvariablen, bestehend aus den Zuständen der Nachbarzellen und $m+1$ globalen Parametern, und werden mit einer bestimmten Wahrscheinlichkeit entsprechend der Funktion p_i für die Bestimmung des Folgezustands ausgewählt. In einem nicht-uniformen PCA haben alle Zellen diese Struktur, jedoch potentiell verschiedene Nachbarschaften N_i, Mengen von Transitionsfunktionen Φ_i und Wahrscheinlichkeitsfunktionen p_i.

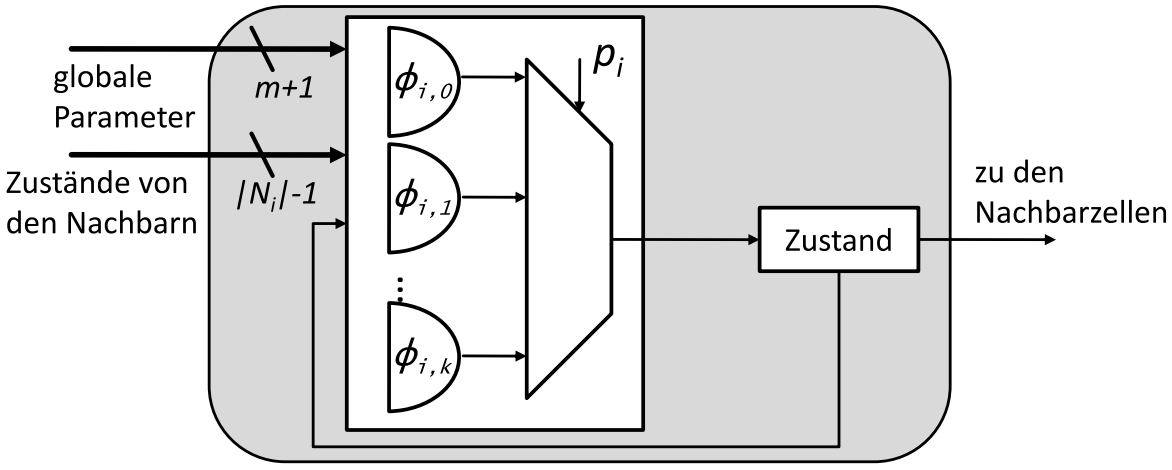

Abb. 2.6.: Struktur einer Zelle im Probabilistischen CA mit globalen Parametern. Die Topologie des Zellularen Feldes, also die Dimension und Nachbarschaft, ist in diesem Schema nicht gezeigt, daher ist auch keine Nicht-Uniformität dargestellt.

Abgesehen von der Transitionsfunktion, die die lokalen Zustandsübergänge der einzelnen Zellen festlegt, gibt es eine weitere die Zustandsübergänge betreffende Kategorisierung und Aufweichung der Definition 2.1. Als *globale Übergangsfunktion* („*global transition function*“) wird in [Cod68, S. 9] die Funktion bezeichnet, die die Transitionsfunktion auf alle Zellen anwendet. In [Hoc98, Hee07] werden CA unter dem Begriff *globales Überführungsschema* in drei Kategorien unterteilt. Ein *synchron-paralleles* Überführungsschema bedeutet, dass die lokale

Transition in allen Zellen zeitgleich geschieht. Das entspricht der Definition 2.1. Der Zustand aller Zellen zu einem bestimmten Zeitpunkt t wird dabei *Generation* genannt[2], die nullte Generation, also die initiale Zuordnung der Zellen zu Zuständen, ist die *Ausgangskonfiguration*. Beim *asynchron-seriellen* Schema werden alle Zellen in einer bestimmten Reihenfolge hintereinander aktualisiert. Das dritte Schema ist die *asynchron-stochastische* Überführung. Dabei werden zu einem Zeitpunkt zufällig Zellen ausgewählt, für die der lokale Zustandsübergang durchgeführt wird.

2.2.2 Eigenschaften von CA-Architekturen

In diesem Abschnitt sollen einige dem CA-Modell zugeschriebenen Eigenschaften beschrieben werden, die für die letztendlich angestrebte Realisierung von konkreten Systemen in Hardware oder Software von Bedeutung sind. Eine solche Realisierung wird *CA-Architektur* genannt.

Ein CA ist *massivparallel*. Das bedeutet, er verfügt über eine sehr große Anzahl an Berechnungseinheiten. In diesem Fall entspricht jede Zelle einer Berechnungseinheit. Diese Berechnungseinheiten sind relativ simpel im Vergleich zu den komplexen (universalen) Prozessoren in Multiprozessorsystemen. Die Einfachheit ist der Tatsache geschuldet, dass die Zellen in jedem Zeitschritt nur eine einzige Berechnung (die Funktion ϕ) durchführen müssen. Im Normalfall sind die Nachbarzellen auch in der geometrischen Interpretation benachbart. Das legt nahe, insbesondere bei Hardwarerealisierungen, Strukturen zu wählen, die ebenso aufgebaut sind. Die Verbindungswege zwischen den Nachbarzellen sind dann weniger aufwendig. Zudem sind die Verbindungen im Vergleich zu Multiprozessorsystemen einfacher, da lediglich der Zustand des Nachbarn übertragen werden muss. Alles in allem hat eine massivparallele CA-Architektur tendenziell eine bessere Skalierbarkeit als andere parallele Systeme mit komplexeren Einheiten, die sie durch die Einfachheit seiner Elemente und seiner Struktur erreicht. Diese Vorteile relativieren sich natürlich, je komplizierter die Regeln und Nachbarschaftsstrukturen im Modell gewählt werden.

In einer CA-Architektur haben die Zellen keinen Schreibzugriff auf ihre Nachbarzellen, sie können lediglich deren Zustände lesen. Das vereinfacht die Realisierung insofern, dass es keine Schreibkonflikte geben kann. Aus dem Modell lässt sich diese Eigenschaft daraus ableiten, dass sich jede Transitionsfunktion immer nur auf eine Zelle bezieht und dass in nicht-uniformen Zellularautomaten jeder Zelle immer nur eine Regel zugeordnet wird.

Spezialrechner, also CA-Architekturen in Hardware, für Zellulare Automaten wurden zuerst von Toffoli und Margolus entwickelt. Deren Cellular Automata Machine (CAM) [Tof84, TM87] ist eine Familie von Rechnern, die bis zur achten Version CAM-8 [Mar93] weiterentwickelt und zum Teil kommerziell verfügbar gemacht wurde. Eine weitere Familie von Spezialarchitekturen zur Zellularverarbeitung ist CEPRA (Cellular Processing Array) aus Darmstadt [HUVW01]. Hardwareimplementierungen, die nicht für beliebige Zellulare Automaten sondern für ganz spezifische CA konstruiert worden sind, gibt es unter anderem in [KF07, FKSL07] zur Verarbeitung von Bildern in Echtzeit mit dem Ziel des Einbaus als Chip in der Kamera. Neben speziellen Architekturen, gibt es auch eine Reihe spezieller Programmiersprachen für die Zellularverarbeitung. CDL (Cellular Description Language) wurde ebenfalls in Darmstadt entwickelt, eine Beschreibung von CDL und eine Auflistung weiterer Sprachen ist in [Hoc98] gegeben.

[2] oft auch als *Konfiguration(t)* bezeichnet

2.2.3 Anwendungen von Zellularen Automaten

Die Anwendungsmöglichkeiten von CA sind wie bei MAS nicht auf ein spezielles wissenschaftliches Feld begrenzt, sondern richten sich nach der Eignung eines Problems, mit einfachen lokalen Regeln ein komplexes globales Verhalten (das ganze Zellulare Feld) beschreiben zu können. Dieses Konzept kommt bei Systemen aus der Natur auch häufig vor. Mit CA können außerdem diskrete raumzeitliche Systeme modelliert werden, und Wolfram schreibt in [Wol83]:

„Cellular automata may thus be considered as discrete idealizations of the partial differential equations often used to describe natural systems.“

Daher ist es nicht verwunderlich, dass insbesondere in den Naturwissenschaften CA eingesetzt werden. Beispielsweise in der Biologie/Medizin zum Modellieren von Evolution in einfachen Ökosystemen, zur Untersuchung von Immunabwehrreaktionen einzelner Körperzellen oder von der Ausbreitung von Seuchen [Bez99], in der Physik zur Modellierung von Partikeldiffusion [Ban99], in der Chemie zur Modellierung des Zusammenwirkens von Stoffen auf molekularer Ebene [KCS00], aber auch in den Politik- und Sozialwissenschaften zur Modellierung von sozialen Netzwerken [BA02]. Nicht zuletzt werden CA auch in der Berechnungstheorie und für mathematische Probleme eingesetzt, z. B. in der Kryptographie [Wol86a]. Die hier genannten Beispiele und Referenzen sind natürlich nur ein kleiner Ausschnitt der tatsächlichen Vielfalt von Anwendungen und sollen veranschaulichen, dass es (analog zu MAS) für CA eine große Diversität von Anwendungsmöglichkeiten gibt. Noch mehr Beispiele für Anwendungen von CA und Referenzen findet man in [GSD+03] oder in [UMN+08].

Das Prinzip, das aus (einfachen) lokalen Regeln (komplexe) globale Effekte entstehen, stellt eine weitere Analogie zu MAS dar, die dazu führt, dass die Anwendungsformen für CA in die gleichen Kategorien eingeordnet werden können wie die Anwendungsformen für MAS (Abschn. 2.1.5): *erklärend*, *vorhersagend*, *problemlösend* und *erzeugend*:

- Zu den erklärenden und vorhersagenden Anwendungen gehören die oben genannten Beispiele aus den Natur- und Sozialwissenschaften. Erwähnenswert, weil häufig verwendet, sind auch Verkehrs- und Fußgängersimulationen (z. B. [Sch08]).

- Problemlösende CA sind u. A. Optimierungsprobleme wie z. B. die Optimierung des Traveling Salesman Problems in [GMW94], Algorithmen zum Finden von kürzesten Wegen [Lee61, Had77, Sou78], die Verarbeitung von Bildern [MRLA82, Ros05] und die Mustererkennung [Rag93]. Unter die problemlösenden CA fallen auch theoretische bzw. künstliche Probleme, wie die räumliche oder zeitliche Synchronisation von Zellen, bekannt als French Flag Problem und Firing Squad Problem [HL73].

- Erzeugende CA sind Modelle zur Erforschung der Entstehung von Komplexität. Ein wichtiges Forschungsfeld ist dabei das *Künstliche Leben* (KL). Bei KL handelt es sich um künstliche Systeme, die (komplexe) Eigenschaften von Lebewesen aufweisen. Eine der bekanntesten Anwendungen in diesem Feld ist John Conway's *Game of Life* [Gar70] (Abschn. 2.3), ein CA, der emergente Eigenschaften aufweist. Aus simplen lokalen Regeln entstehen komplexe globale Muster. Weitere Beispiele für KL mit CA findet man in [Lan86]. KL ist aber nicht nur auf das CA-Modell beschränkt. [Lev93, Ada98] sind Einführungen in das Thema allgemein.

In [Hoc98, S. 14f.] werden ebenfalls vier Anwendungsformen unter anderen Bezeichnungen unterschieden: theoretische Modelle (erzeugend und problemlösend), Simulationsmodelle mit bekannten lokalen Gesetzmäßigkeiten (vorhersagend), Simulationsmodelle mit bekanntem globalem Systemverlauf (erklärend) und Endwertprobleme (problemlösend). Zusätzlich unterscheidet Hochberger makroskopische und mikroskopische Simulationsmodelle und bezieht sich auf die reale Größe der Einheiten, die die Zellen repräsentieren. Zu jeder Kategorie wird dort eine Reihe von Anwendungen aufgezählt und informell beschrieben.

2.3 Multiagentensysteme mit Zellularen Automaten

Einige Analogien zwischen den Eigenschaften von MAS und CA sind in den letzten beiden Abschnitten bereits diskutiert worden. Darunter fällt das emergente Verhalten der Systeme und die sich überschneidenden Anwendungsgebiete. In der Literatur findet man unterschiedliche Formen des Zusammenhangs von CA und MAS:

a) CA als ein Spezialfall von MAS. Wenn man die Zellen als Agenten interpretiert (Abb. 2.7(a)), erhält man ein System von autonomen, wahrnehmenden, reaktiven Agenten. Diese Agenten sind nicht mobil und je nach Verwendung von uniformen oder nichtuniformen Regeln haben alle Agenten das gleiche oder ein unterschiedliches Verhalten. Außerdem verhalten sie sich je nach globaler Überführungsfunktion asynchron oder synchron [BMS02]. Bei dieser Sichtweise ist jeder CA ein MAS, jedoch können komplexere MAS nur mit Einschränkungen im CA modelliert werden. Zu den eben genannten Einschränkungen kommen noch die statische und lokale Nachbarschaft der Zellen und somit die eingeschränkte Sicht der Agenten auf ihre Umwelt, sowie die zeitliche und räumliche Diskretheit.

b) Ein MAS als Mehrschichtenmodell, das einen CA als unterliegende Ebene zur Modellierung einer räumlichen Umwelt oder eines Teils der Umwelt (explizit ohne Agenten) benutzt (Abb. 2.7(b)). Diese Form der Modellierung wird vor allem durch zwei Eigenschaften des CA motiviert. Zum einen die Immobilität der Zellen (als Agenten) und zum anderen die Synchronität der Agenten [CW06].

c) Ein MAS, das in einem CA modelliert wird, der durch Erweiterung des Modells oder durch geschickte Definition von Regeln nicht mehr den Beschränkungen aus a) unterliegt, insbesondere der Immobilität. D. h., dass die Agenten zwar in den Zellen (deren Zustände und Regeln) integriert sind, aber nicht den Zellen entsprechen (Abb. 2.7(c)). Gruner beschreibt z. B. in [Gru09], dass ein generalisierter CA (mit irregulärer Netztopologie) in ein MAS umgewandelt werden kann und umgekehrt.

Im den folgenden Teilabschnitten sollen diese Zusammenhänge anhand von konkreten Beispielen dargelegt werden. Fall a) in Abschn. 2.3.1, Fall b) in Abschn. 2.3.2 und Fall c) in Abschn. 2.3.3.

2.3.1 CA als Spezialfall von MAS mit Einschränkungen

Das erste Beispiel ist Conway's *Game of Life*, beschrieben in [Gar70]. Es handelt sich dabei um einen uniformen CA mit Dimension $d = 2$, dem Zustandsalphabet $S = \{0, 1\}$ und der Moore-

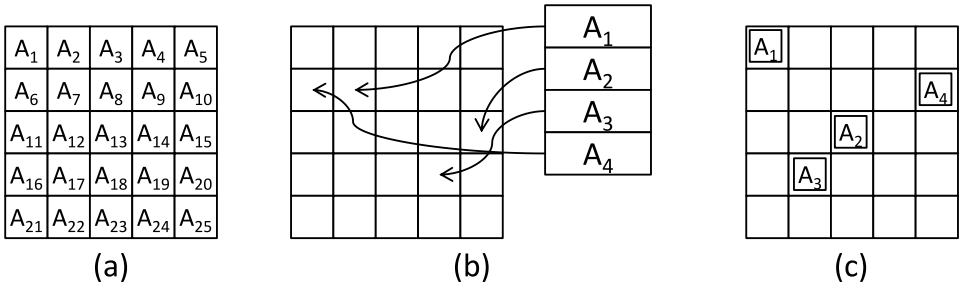

Abb. 2.7.: Formen des Zusammenhangs von MAS und CA, (a) jede Zelle wird als Agent (A) interpretiert, (b) die Agenten benutzen den CA als räumliche Umgebung, sind aber nicht in den Zellen enthalten, (c) jeder Agent ist in einer Zelle gespeichert, aber die Zelle besteht aus mehr als nur dem Agenten.

Nachbarschaft mit Radius $r = 1$. Die Zustände 0 und 1 werden dabei auch als „lebendig“ und „tot“ bezeichnet. Es gibt also lebendige und tote Zellen, daher der Name „Game of Life“. Die Transitionsfunktion lässt sich informal wie folgt beschreiben.

- Eine Zelle nimmt zum Zeitpunkt $t+1$ den Zustand lebendig an, wenn exakt drei Nachbarzellen (die Zelle selbst ausgeschlossen) zum Zeitpunkt t lebendig sind.

- Eine Zelle behält ihren Zustand unverändert bei, wenn exakt zwei Nachbarzellen lebendig sind.

- In allen anderen Fällen ist die Zelle zum Zeitpunkt $t+1$ tot.

Diese einfachen Regeln erzeugen aus unterschiedlichen Ausgangskonfigurationen deterministische, aber irreduzible globale Muster auf dem Zellularen Feld. Darunter sind statische Muster, über Generationen hinweg oszillierende Muster und sogenannte Gleiter: Formationen von lebendigen Zellen, die sich mit Fortschreiten der Generationen zu bewegen scheinen, obwohl sich diese Bewegung eigentlich durch das „sterben“ einer Zelle und das „geboren werden“ der Nachbarzelle ergibt. Eine schier unglaubliche Fülle von verschiedenen auftretenden Strukturen sind dabei in unzähligen Arbeiten „entdeckt“ worden [Gar83, S. 241][Lev93, S. 65ff.]. Zudem kann dieser CA als universelles Berechnungsmodell klassifiziert werden. Zu zeigen, dass dies mit sehr simplen Regeln und wenigen Zuständen möglich ist, (im Gegensatz zu von Neumann's komplexem CA) war die ursprüngliche Absicht von Conway [Lev93, S. 7]. Als MAS lässt sich dieser CA wie folgt interpretieren: Die Agentenwelt entspricht dem Zellularen Feld, die Agenten entsprechen den Zellen. Die zu einem bestimmten Zeitpunkt wahrnehmbare Umwelt der Agenten entspricht der Nachbarschaft exklusive der eigenen Zelle, während die zu einem bestimmten Zeitpunkt modifizierbare Umwelt der Agenten leer ist. Modifiziert werden kann im CA nur die eigene Zelle, die aber nicht zur Umwelt gehört. Das Verhalten der Agenten ist durch die Transitionsfunktion gegeben. Agenten sind autonom, wahrnehmend, reaktiv, kommunikativ und abgeschlossen. Ein globales Ziel verfolgen sie nicht. Dieses System ist also ein erzeugendes (s. auch Abschn. 2.2.3). Die Umwelt der Agenten lässt sich als deterministisch, statisch, eingeschränkt beobachtbar und diskret klassifizieren.

2.3.2 MAS als Mehrschichtenmodell mit CA als Umweltschicht

Mehrschichtenmodelle für MAS mit einer CA-Schicht als Modell der Umwelt ohne Agenten, in der sich die Agenten bewegen, gibt es in unterschiedlichen Szenarien. Allen gemein ist jedoch, dass der CA als diskretes Modell einer räumlichen Umwelt eingesetzt wird. Die Umwelt (aus Sicht des Agenten) besteht hier also aus dem gesamten Zellfeld und zusätzlich aus den restlichen Agenten. Welcher Teil davon modifizierbar und wahrnehmbar ist, hängt von der jeweiligen Anwendung ab. Zwei Beispiele sollen im Folgenden diese Vorgehensweise beim Modellieren von MAS beleuchten.

In [CGHH89] wird mit der *Phoenix* Testumgebung die Entwicklung von Waldbränden in einer CA-Schicht beschrieben, der eine andere Schicht zur Simulation des CA unterliegt. In zwei über dem CA liegenden Schichten wird das Verhalten der Agenten (Einheiten zur Feuerbekämpfung) und deren Kommunikation untereinander simuliert. Eine geeignete Schnittstelle, die die Agentenaktionen in die diskrete, synchrone CA-Schicht „übersetzt“, sorgt dafür, dass die Aktionen einen Effekt in der CA-Schicht haben.

Ein Modell zur Simulation künstlicher Gesellschaften wird in [EA96] vorgestellt. Es soll damit untersucht werden, wie aus dem Verhalten Einzelner und der Interaktion zwischen Individuen soziale Strukturen und Gruppenverhalten entstehen. Im *Sugarscape*-Modell werden die Agenten separat von der Umwelt in der sie operieren und mit der sie interagieren betrachtet. Ein CA wird dafür als in ein Gitter aufgeteilter Raum ohne Agenten eingesetzt. Zusätzlich wird ein Modell von adaptiven, beweglichen Agenten (ohne zugrunde liegenden Raum) verwendet. Das Sugarscape-Modell wird als Synthese von CA und Agenten betrachtet. In der einfachsten Sugarscapevariante bewegen sich Agenten im zweidimensionalen Raum (kartesische, ganzzahlige Koordinaten) und konsumieren Zucker aus der Umwelt. Diese Umwelt ist ein zweidimensionales CA-Gitter und beinhaltet in jeder Zelle eine bestimmte Menge an Zucker und einen Maximalwert, den diese Menge nicht überschreiten darf. Auf diese Weise ist das Zustandsalphabet festgelegt. Der Zuckerwert wird nach einer variablen Wachstumsregel (der Zellregel) nach dem Konsum durch die Agenten wieder aufgefüllt. Die Wachstumsregel kann als Transitionsfunktion angesehen werden. Die Nachbarschaft in diesem einfachen Modell besteht nur aus der Zelle selbst, auf die sie sich bezieht. Die Agenten sind in diesem Fall aus Sicht des CA ein externer Einfluss, der unabhängig von der Transitionsfunktion den Zustand der Zellen beeinflussen kann. Schrittweise wird dann im Buch von Epstein und Axtell das Sugarscape-Modell weiterentwickelt und kompliziertere Regeln verwendet. Der CA und die Agenten bleiben jedoch weiterhin separat.

2.3.3 MAS in (erweiterten) CA ohne Einschränkungen

Von den oben genannten Einschränkungen für als CA modellierte MAS sind einige relativ leicht zu umgehen, wenn man die in Abschn. 2.2.1 Varianten implementiert. So können z. B. MAS mit inhomogenen Agenten, also verschiedenen Typen von Agenten, einfach mit nicht-uniformen CA modelliert werden. Agenten, die sich nicht synchron verhalten, können über ein asynchron-serielles globales Überführungsschema modelliert werden. Die statische (unveränderliche) Sicht der Agenten auf ihre Umwelt, die durch die statische Nachbarschaft entsteht, kann durch die Verwendung von GCA umgangen werden. Es bleibt also noch die Beweglichkeit der Agenten übrig, die im Modell nicht verankert ist, denn offensichtlich kann hier nicht mehr die Zelle als

Agent betrachtet werden. Stattdessen muss eine Form gefunden werden, in welcher im Zellzustand alle notwendigen Informationen, die den Agenten und seine Eigenschaften ausmachen, enthalten sind. Dann allerdings ergibt sich für die Bewegung das Problem, dass im Sinne einer CA-Architektur keine Schreibzugriffe, also das Kopieren eines Agenten von einer in die andere Zelle, möglich sind. Bei einer etwas allgemeineren Betrachtungsweise sind nicht nur Bewegungen des Agenten betroffen, sondern alle Aktionen, die auf einen Zustand in einer Nachbarzelle wirken sollen. Diese Schreibzugriffe müssen durch entsprechende Zellregeln emuliert werden.

Zunächst soll der Fall betrachtet werden, in dem sich ein Agent von einer Zelle in eine Nachbarzelle bewegt. Das bedeutet, der Agent befindet sich zum Zeitpunkt t nur in einer Quellzelle z_1, zum Zeitpunkt $t+1$ soll er sich nur in der Zielzelle z_2 befinden. Es muss also eine Änderung des Zustands in Zelle z_1 und gleichzeitig eine Änderung des Zustands in Zelle z_2 erfolgen. Nur bei Agentensystemen, die das Klonen und Löschen von Agenten erlauben ergibt sich dieses Problem je nach Design des MAS nicht. Z. B. können die *Marching Pixels*, die zur Bildverarbeitung eingesetzt werden, sterben oder sich vereinen [Kom10]. Da der Agent sich nicht selbst in die Zelle z_2 kopieren kann, muss es eine entsprechende Regel in Zelle z_2 geben, die den Agenten aus der Zelle z_1 kopiert. In einer räumlichen Vorstellung könnte man davon sprechen, dass der Agent, statt sich zu bewegen, von seiner Umwelt über das Zellulare Feld gezogen wird.

Nun sollen einige Ansätze anhand von Beispielen aus der Literatur vorgestellt werden, die es erlauben, ein komplettes MAS innerhalb des CA-Modells zu beschreiben, in dem die Agenten beweglich sind. Dabei sollen auch die Modellierung von Bewegungsgeschwindigkeit und räumliche Konflikte berücksichtigt werden. Räumliche Konflikte ergeben sich dann, wenn sich mehr als ein Agent auf die gleiche Zelle bewegen will. Das Grundproblem bei einem synchronparallelen CA ist dabei, dass sich die Agenten unter Umständen nicht in ihrer gegenseitigen Nachbarschaft befinden, sie sich also nicht wahrnehmen können und daher versuchen könnten, sich auf dieselbe Zielzelle zu bewegen. Man könnte die Zellen auch so entwerfen, dass sich eine endliche Menge an Agenten gleichzeitig auf einer Zelle befinden darf, dann entsteht dennoch der Konflikt, sobald diese Menge überschritten wird. Unendlich viele Agenten können nicht zugelassen werden, da jede Zelle die Ressourcen für alle möglicherweise auf ihr befindlichen Agenten bereitstellen muss.

Spicher et al. präsentieren in [SFS09] den *Transactional CA*, der sich aus der Übersetzung eines diskreten, in einer räumlichen Welt befindlichen MAS in das CA-Modell ergibt. Besonderes Augenmerk wird dabei auf die Aktualisierung der Zustände gelegt, da z. B. die Reihenfolge bei einer seriellen Aktualisierungsmethode das Ergebnis einer Simulation beeinflussen kann. Eine sequentielle Aktualisierung und damit eine sequentielle Bewegung der Agenten (also ein Agent nach dem anderen) ist eine Methode, um Konflikte sich bewegender Agenten zu verhindern. Die Autoren geben zwei Gründe an, warum man dennoch eine synchrone Aktualisierung vornehmen sollte. Zum einen sei eine Implementierung eines dann notwendigen Schedulers, der die Reihenfolge steuert, auf einem parallel arbeitendem Gerät schwierig. Mit dem Gerät ist eine Hardwarerealisierung einer CA-Architektur gemeint. Zum anderen führe die Einführung eines Schedulers zu einem externen Einfluss, der nicht in der eigentlichen Problemformulierung vorgesehen war. Für die Bewegung eines Agenten von einer Quellzelle z_1 auf eine Zielzelle z_2, die eine Konfliktbehandlung einschließt, wird ein Prozess mit drei Schritten vorgeschlagen:

1. *Request*: Quellzelle z_1 ändert ihren Zustand so, dass die Zielzelle z_2 im nächsten Schritt sehen kann, ob ein Bewegungswunsch besteht.

2. *Approval-Rejection*: Zielzelle z_2 überprüft alle Bewegungswünsche der Nachbarn. Sie ändert ihren Zustand, um so den Nachbarn zu signalisieren, ob eine Bewegung stattfinden kann oder nicht. Sie kann dabei entweder ein Exklusivrecht für einen Agenten erteilen oder bei mehreren Anfragen, allen Agenten die Bewegung untersagen.

3. *Transaction*: Je nach Zustand von Zielzelle z_2, können nun z_2 und z_1 eine konsistente Entscheidung treffen und ihre Zustände entsprechend ändern.

Eine fast identische Konfliktlösungsstrategie wird schon früher in [HHW99] präsentiert. Dort werden nur zwei *Phasen* statt drei Schritte unterschieden, weil der Request-Schritt als impliziter Zustand des Agenten gesehen wird. Der Agent hat also eine permanente (aber veränderliche) Bewegungsrichtung. Die erste Phase entspricht dem Approval-Rejection-Schritt und wird *Conflict-Resolution*-Phase genannt. In der *Movement*-Phase wird dann wie im Transaction-Schritt die eigentliche Transaktion vollzogen (durch kopieren und löschen). Beim Kopieren wird bereits die Bewegungsrichtung des Agenten mit kopiert und kann je nach Regel auch verändert werden.

In [HHB06] wird eine Möglichkeit zur Konfliktlösung in nur einer Phase beschrieben. Dabei bedient sich die Zielzelle einer asynchronen Feedback-Logik, die aus der Bewegungsrichtung des Agenten innerhalb einer diskreten Zeitstufe ein Ablehnungs- oder Bestätigungssignal generiert. Dieses generierte Signal ist also zum Zeitpunkt t schon in der Quellzelle abrufbar. Diese Lösung ist allerdings nur auf einer das Modell konkretisierenden Architektur-Ebene implementierbar.

Die bisher genannten Möglichkeiten für die Konfliktlösung benutzen die Zielzelle als „Ersatzauge“ des Agenten, welches für den Agenten die Informationen sammelt. Man kann das Problem aber auch an der Ursache packen und den Agenten eine erweiterte Sicht geben, d. h. man erweitert die Nachbarschaft soweit, dass alle notwendigen Informationen von der Quellzelle aus gelesen werden können. Bei einem MAS, in dem die Agenten sich orthogonal auf dem Zellfeld bewegen können, reicht eine von Neumann-Nachbarschaft mit Radius 2 aus, um eine Transitionsregel zu erstellen, die weder von einer asynchronen Feedback-Logik abhängt, noch mehr als einen Zeitschritt benötigt, um eine Bewegung durchzuführen. Das ist der Ansatz, der auch in dieser Arbeit verwendet wird.

Toffoli und Margolus beschreiben in [TM87, Kap. 12] die *Margolus-Nachbarschaft*, die eine andere Art der Nachbarschaftserweiterung darstellt. Unter dem Begriff *Partitioning Cellular Automaton* fassen sie endliche Blöcke von Zellen zusammen, die sich nicht überschneiden und regulär im Zellularen Feld angeordnet sind. Für jeden der Blöcke existiert eine Blockregel, die die Zustände innerhalb des Blocks aktualisiert. In jedem Zeitschritt wird die Aufteilung des Zellularen Felds wieder verändert, so dass andere Mengen von Zellen zusammen in einem Block zusammengefasst werden. Auf diese Weise können bewegliche Teile (Agenten) nie in Konflikte geraten, da ja immer nur Blöcke als Ganzes aktualisiert werden und diese nicht mit anderen Blöcken in Berührung kommen. Durch die Veränderung der Zuordnung zu den Blöcken, können Agenten beliebige Positionen erreichen.

Bisher noch nicht diskutiert wurde der Fall, dass Agenten in einem MAS unterschiedliche Bewegungsgeschwindigkeiten haben können. Eine höhere Geschwindigkeit als die Bewegung von einer Zelle zu einer anderen würde bedeuten, dass ein Agent in einem Zeitschritt gleich mehrere Nachbarzellen überspringt. Eine geringere Geschwindigkeit würde bedeuten, dass ein Agent mehr als einen Zeitschritt benötigt, um in die Nachbarzelle zu gelangen.

Dijkstra et al. stellen in [DJT01] eine Lösung vor, die unterschiedliche Geschwindigkeiten modelliert, indem die Zeitschritte abhängig von der Maximalgeschwindigkeit der Agenten in Teilschritte („time-step-slices“) aufgespalten werden. Agenten, die die Maximalgeschwindigkeit

haben, bewegen sich dann in jedem Zeitschritt eine Zelle weiter, langsamere Agenten fügen Warteschritte ein.

Eine höhere Geschwindigkeit kann aber auch über eine erweiterte Nachbarschaft realisiert werden. Dabei muss ein Agent auf alle Zellen „im Blick“ haben, die von der Zielzelle aus gesehen bei Maximalgeschwindigkeit in deren Reichweite liegen. Das führt bei hohen Geschwindigkeiten zu großen Nachbarschaften und damit zu komplexeren Zellregeln. Eleganter kann das Problem der großen Distanzen mit GCA gelöst werden. In [SHH10] wird eine GCA-Architektur entwickelt, um Verkehrssimulationen mit Agenten, die eine bestimmte Maximalgeschwindigkeit haben, effizienter machen zu können.

2.4 Kapitelzusammenfassung

In diesem Kapitel wurden die Grundlagen von MAS beschrieben. Begriffe zur Beschreibung der Charakteristika von Agenten und MAS wurden definiert, und Klassifikationen von MAS vorgestellt, um darauf aufbauend eine Modellierung des Systems im nächsten Kapitel vornehmen zu können. Einige grundlegenden Eigenschaften von MAS, wie *Emergenz* und *Selbst-X-Eigenschaften* wurden erklärt und auf den Zusammenhang von lokalem Verhalten der Agenten und globalem Verhalten des Gesamtsystems hingewiesen. Dieser Zusammenhang ist von entscheidender Bedeutung bei der Optimierung des MAS, denn die Angriffspunkte des Entwicklers mit einem Optimierungsverfahren liegen immer im lokalen Verhalten, obwohl es das globale Verhalten ist, das letztendlich optimiert werden soll.

Des Weiteren wurden die Grundlagen von Zellularautomaten auf mathematischer Basis vorgestellt und einige prinzipielle Möglichkeiten der Nutzung des CA-Modells als MAS beschrieben. Insbesondere auf die Problematik der Modellierung von beweglichen Agenten und auf aus der Literatur bekannte Ansätze zur Überwindung dieser wurde hingewiesen. Diese Ansätze können nun bei der Entwicklung eines allgemeinen Modells in Kap. 3 für MAS mit beweglichen Agenten aufgegriffen und gegebenenfalls erweitert werden.

3 Modellierung von Multiagentensystemen in Zellularen Automaten

In Kap. 2 wurden einige spezielle Beispiele von MAS, die als CA modelliert sind, genannt. Auf Einschränkungen und Möglichkeiten diese aufzulösen wurde hingewiesen. In diesem Kapitel wird nun ein allgemeines Modellierungskonzept von MAS in CA vorgestellt, das anpassbar und erweiterbar für spezifische MAS ist. Unter die Anpassbarkeit fallen Bereiche wie die Menge der Aktionen der Agenten, die Modellierung des Verhaltens der Agenten, deren Sensorik, die Struktur der Zellen im Sinne von räumlicher Anordnung und die Typen von Objekten in der Welt des MAS.

In Abschn. 3.1 wird in Erweiterung zu Abschn. 2.3 die allgemeine Vorgehensweise beim Modellieren eines MAS mit beweglichen Agenten in CA beschrieben. Das Resultat ist ein parametrisierbares MAS, das sich für spezielle Anwendungen anpassen lässt und vom Agentenverhalten abstrahiert. Es ist somit für eine Vielzahl der Formen von künstlicher Intelligenz geeignet.

In Abschn. 3.2 wird die Modellierung des Agentenverhaltens mit endlichen Zustandsautomaten besprochen und deren Einbettung in das CA-Modell detailliert beschrieben. Danach werden alternative Möglichkeiten zur Modellierung des Verhaltens des Agenten besprochen, die ebenfalls in das allgemeine CA-Modell integriert werden könnten. Insgesamt wird damit das erste Teilziel, der Entwurf eines Modells zur Beschreibung von Multiagentensystemen in Zellularen Automaten, erreicht.

3.1 Modellierungskonzept von MAS in CA

Ausgehend von der Diskussion von MAS und CA in Abschn. 2.3 wird die Beschreibung eines als CA modellierten MAS konkretisiert und um alle Elemente erweitert, die für die vollständige Beschreibung eines MAS notwendig sind. Dabei wird zunächst das Verhalten der Agenten außen vor gelassen, um ein allgemeines Modell zu erhalten, das für jegliche Form von perzeptiven und reaktiven Agenten geeignet ist. Bei konkreten MAS mit einer endlichen Anzahl von Objekten und bekannten Regeln kann das allgemeine Modell unter Umständen stark vereinfacht werden. Grundsätzlich sollte man außerdem erwähnen, dass die im Folgenden beschriebene Modellierung nicht die einzig mögliche ist.

Um eine vollständige Beschreibung eines MAS zu erhalten, müssen alle Komponenten der Agentenwelt und ihre Attribute angegeben werden. Die Komponenten der Agentenwelt sind *Objekte* verschiedenen Typs (*Objekttyp*), die in zwei Kategorien aufgeteilt werden können:

1. *Agenten* (möglicherweise verschiedenen Typs). Agenten sind spezielle *aktive Objekte*, die autonom, perzeptiv und reaktiv sind. Durch die Autonomität und Aktivität wäre aus sprachlicher Sicht sicherlich der Begriff „Subjekt“ angebrachter. Das Wort Objekt soll aber hier gerade nicht im Sinne von „passiv“ verstanden werden, sondern als ein in sich abgeschlossener Bestandteil des Gesamtsystems.

2. *passive Objekte*, die nicht alle erforderlichen Kriterien erfüllen, um als Agent bezeichnet werden zu können. Dabei kann es sich z. B. um gegenständliche Objekte wie Hindernisse oder Ressourcen, die von Agenten eingesammelt werden, handeln, aber auch um abstrakte Objekte, wie Informationen, die von den Agenten gelesen und/oder verändert werden können.

Die Beweglichkeit eines Objekts ist weder eine hinreichende, noch eine notwendige Bedingung für die Einordnung als Agent. Ebenso verhält es sich mit den Eigenschaften modifizierend und kommunikativ. Einige Attribute aus Tab. 2.1 können lediglich Agenten betreffen, da sie eine Autonomität und Reaktivität voraussetzen (z. B. Rationalität und Lernfähigkeit). Passive Objekte können auch dynamisch sein, d. h. sie können erzeugt werden, gelöscht werden, kopiert werden oder sich vereinigen. Dies kann entweder durch Aktionen von Agenten verursacht werden oder aber durch Beeinflussung von anderen Objekten oder von globalen Parametern, wie der Zeit. Jedes Objekt im MAS kann *Attribute* besitzen, die fixe oder veränderliche Werte annehmen können. Wie die dynamischen Objekte, können Werte entweder durch modifizierende Agenten, andere Objekte oder globale Parameter verändert werden. Ein Beispiel für dynamische passive Objekte mit veränderlichen Werten ist das Objekt vom Typ *Pheromon* mit dem Attribut *Intensität*. Nehmen wir ein MAS an, in dem Agenten Objekte des Typs Pheromon erzeugen. Direkt nach dem Erzeugen haben Pheromone die Intensität 10. In jedem Zeitschritt, soll dann die Intensität um 1 dekrementiert werden. Die Dynamik entsteht also durch die Aktion der Agenten, die Werte des Attributs Intensität werden in Abhängigkeit vom globalen Parameter Zeit verändert. Pheromone sind keine Agenten, da sie nicht perzeptiv und reaktiv sind.

3.1.1 Das Zellulare Feld als Agentenwelt

Jedes Objekt hat zu jeder bestimmten Zeit eine bestimmte Position in der Agentenwelt. Dies kann eine Position in kartesischen Koordinaten sein oder eine Position in einer Topologie, die keine räumliche Interpretation besitzt. In beiden Fällen wird diese Position jedoch durch genau eine Zelle eines Zellularen Feldes repräsentiert. Man könnte auch sagen, jedes Objekt befindet sich in oder auf genau einer Zelle des Zellularen Feldes. Das Objekt mitsamt seinen Attributen ist also in der Zelle gespeichert. Die Position selbst ist nicht zwangsläufig ein zusätzliches Attribut, sondern implizit durch die Identifikation der Zelle (und ihrer Position innerhalb des Zellularen Feldes) gegeben. In MAS, in denen die Position eines Objektes von Agenten aus der Nachbarschaft gelesen werden soll, oder die eigene Position bekannt sein soll, muss die Position als Attribut vorliegen.

Auf jeder Zelle können potentiell mehrere Objekte gleichen oder verschiedenen Typs gleichzeitig sein. Wie viele Objekte jeden Typs gleichzeitig auf einer Zelle sein können, wird durch die konkrete Realisierung der *Zellstruktur* beschränkt. Jede Zelle muss alle Ressourcen für alle Objekte, die im Verlauf einer Simulation eines MAS auf dieser Zelle positioniert sein könnten, bereitstellen, d. h. alle Attribute aller Objekttypen müssen (gegebenenfalls mehrfach) gespeichert sein. Zusätzlich muss die Zelle die Regeln beinhalten, die für die jeweiligen Objekttypen gelten. Angenommen, eine Zelle darf maximal ein Objekt (gleich welchen Typs) beherbergen, so muss sie dennoch für alle im MAS befindlichen Typen, jeweils einmal alle Attribute speichern. Um nun festzustellen, ob ein Objekt, und wenn ja, welches Objekt sich gerade auf der Zelle befindet, ist ein weiteres Merkmal notwendig: Der *Zelltyp* τ legt in jeder Zelle fest, welche Objekte sich gerade auf bzw. in ihr befinden, also welche von den gespeicherten Attributen relevant sind.

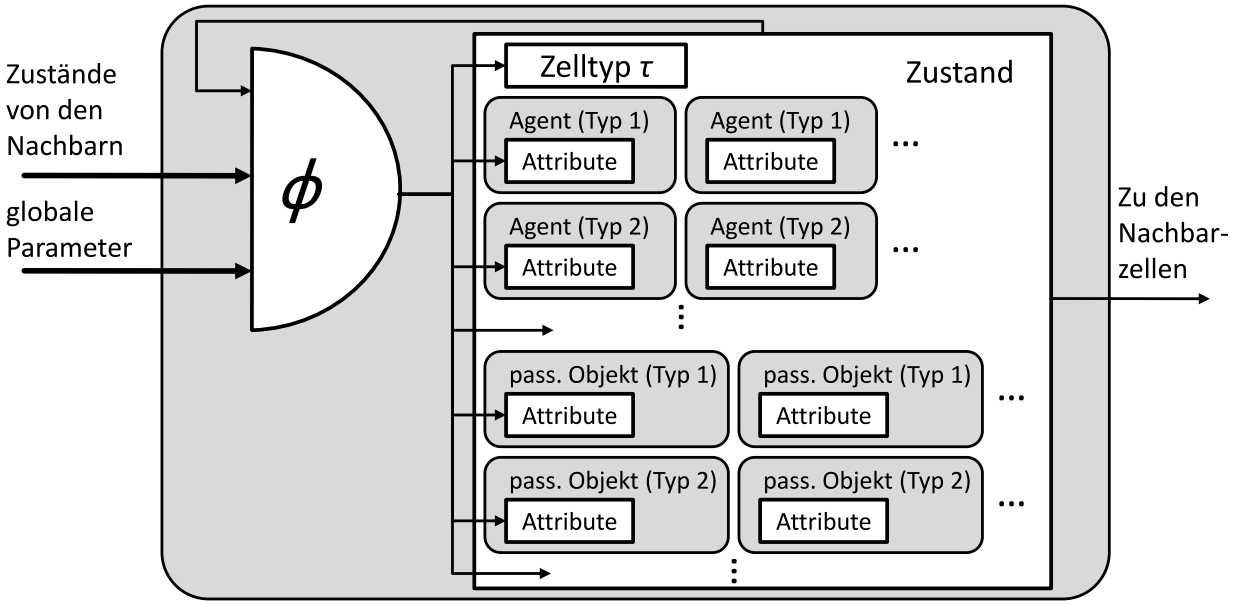

Abb. 3.1.: Allgemeine Struktur einer Zelle in einem CA-Multiagentensystem. Es ist für jeden Objekttyp für die im System maximal erlaubte Anzahl (Anzahl der Spalten) Objekte jeden Typs (Anzahl der Zeilen), ein Speicherplatz vorgesehen. Darin sind alle Attribute für das Objekt enthalten.

Abb. 3.1 zeigt den allgemeinen Aufbau der Zellstruktur, die eine Konkretisierung der Zellstruktur aus Abb. 2.6 darstellt, allerdings der Übersicht halber ohne die Berücksichtigung von probabilistischen Regeln. Der Zustand einer Zelle ist die Gesamtheit aus Zelltyp und allen Attributen. Die Regeln für passive Objekte sind in der Transitionsfunktion ϕ ebenso enthalten wie die Regeln für aktive Objekte. Im Fall der Agenten werden die Regeln auch als Agentenprogramm bezeichnet und beinhalten eine Form von Intelligenz, die Entscheidungen trifft (s. Abschn. 3.2). Die Zelltypen und alle Attribute der Nachbarzellen können als Variablen in die Regeln und Agentenprogramme eingehen. Die Regeln der passiven Objekte bzw. das Programm der Agenten werden nur dann sinnvoll angewendet, wenn die betreffenden Objekte tatsächlich an der Zellposition sind. Der Zelltyp liefert also die Grundlage einer Zellregel mit Fallunterscheidung. Ein Teil der Zellregel ist natürlich auch die Aktualisierung des Zelltyps, wenn Objekte gelöscht, verschoben, vereinigt oder erzeugt werden. Der Zelltyp einer Zelle ist also nicht statisch, sondern verändert sich durch die Zellregel, die wiederum vom Zelltyp abhängt.

Es ist nicht einfach möglich, jede Komponente dieser Struktur, genau einer der Komponenten des MAS zuzuordnen, wie sie in Abschn. 2.1 beschrieben und in Abb. 2.1 sowie Abb. 2.2 dargestellt sind. Dennoch entspricht diese Struktur immer noch dem allgemeinen Konzept eines MAS. Der Datenzustand und der evtl. vorhandene Kontrollzustand sind in den Attributen eines Agenten codiert. Der Sensor, die Kontrollfunktion und der Aktuator sind allesamt in der Transitionsfunktion ϕ enthalten. Das bedeutet, dass ϕ für jeden Agenten in der Zelle diese drei Komponenten beinhaltet. Demzufolge sind die Wahrnehmung und die Entscheidung hier nicht sichtbar. Die Aktion eines Agenten ist hier ein bestimmter Anteil des Outputs der Funktion ϕ, lässt sich aber auch nicht unbedingt daraus extrahieren, da der Output gleich mehreren Aktionen von mehreren Agenten entspricht. Die Umwelt eines Agenten setzt sich zusammen aus der Gesamtheit der Zustände aller anderen Zellen, dem Zelltyp der eigenen Zelle sowie allen Attributen der anderen Objekte in der eigenen Zelle. Der für den Agenten wahrnehmba-

re Ausschnitt der Umwelt ist abhängig von der konkreten Realisierung der Funktion ϕ, bleibt jedoch beschränkt auf die Nachbarschaft der Zelle. Abb. 3.2 zeigt dies für die von Neumann-Nachbarschaft mit Radius 1. Der modifizierbare Ausschnitt der Umwelt bleibt beschränkt auf den Teil der Umwelt, der sich auf der eigenen Zelle befindet, weil nur dort Schreibzugriffe möglich sind.

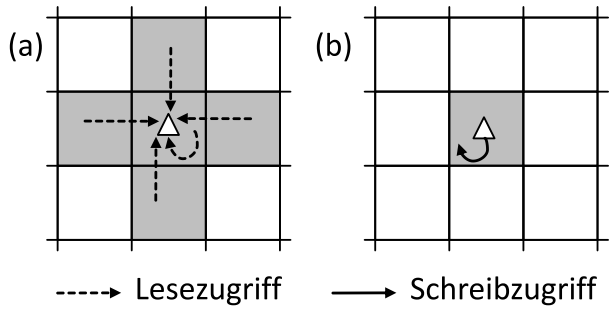

Abb. 3.2.: (a) Wahrnehmbare und (b) modifizierbare Ausschnitte der Umwelt aus Sicht eines Agenten (als Dreieck dargestellt) in der mittleren Zelle.

In den folgenden Abschnitten wird die Struktur weiter „entfächert“, so dass die einzelnen Komponenten besser sichtbar werden. Außerdem wird eine Erweiterung eingeführt, die durch Emulation von Schreibzugriffen den modifizierbaren Ausschnitt der Umwelt potentiell auf die ganze Nachbarschaft ausweitet.

Es ist nicht unbedingt erforderlich, dass alle Zellen die gleiche Struktur besitzen. So können die Nachbarschaft, die Anzahl und Art der möglichen enthaltenen Objekte und deren Regeln variieren. Dies entspricht der Verwendung unterschiedlicher Regelmengen, die auch probabilistisch sein können, und einer Zuordnungsfunktion zu den Zellen, wie in Definition 2.2. Über die Angabe des Quadrupels, welches einen nicht-uniformen CA definiert, kann also die Struktur jeder einzelnen Zelle, sowie die Topologie des Feldes und die Dimension angegeben werden. Implizit hat man damit auch schon die Randbehandlung über spezielle Regeln oder einen Wrap-around gemacht. Beide Möglichkeiten bedeuten gleichzeitig eine endliche Begrenzung des Zellularen Feldes. Eine notwendige Angabe zur vollständigen Spezifizierung eines MAS ist dann auch die Größe (Länge in jeder Dimension) des Feldes.

Zum Simulieren des MAS braucht man schließlich noch eine Ausgangskonfiguration. Diese ergibt sich aus den initialen Positionen aller Objekte, d. h. aus den Zelltypen aller Zellen, und den initialen Werten aller Objektattribute.

3.1.2 Interaktionen der Objekte

Nun soll die Transitionsfunktion mit den Regeln aller aktiven und passiven Objekte etwas detaillierter untersucht werden. Bisher enthält die Funktion ϕ die Regeln aller Objekte. Sie bildet von der Menge S^N auf die Menge S ab, aber alle Elemente von S sind selbst schon Tupel aus je einem Wert für jedes Attribut von jedem Objekt plus einem Wert für den Zelltyp. Diese Komplexität soll durch die Aufspaltung der Transitionsfunktion in mehrere Regeln (eine für jedes Objekt) vereinfacht werden. Abgesehen von der einfacheren Modellierung, entspricht diese Vorgehensweise auch eher der Vorstellung, dass jedes Objekt für sich eine Transitionsfunktion besitzt. Dabei ist

aber auch zu beachten, dass eventuell Abhängigkeiten zwischen den Transitionen der einzelnen Objekte bestehen.

Im Folgenden wird für die Beschreibung der Struktur vom allgemeinen Fall möglichst komplexer Objekte ausgegangen. Passive Objekte, sind solche, denen etwas fehlt, um ein Agent zu sein. Daher kann man in der Zellstruktur prinzipiell davon ausgehen, dass alle Objekte wie Agenten strukturiert sind, d. h. alle Komponenten des Agenten bereitstellen. Wenn diese nicht benötigt werden, können sie in einer konkreten Implementierung weggelassen werden.

Ein Objekt ist *modifizierbar*, wenn es ein Attribut besitzt, dass verschiedene Werte annehmen kann. Die Modifikation kann durch das Objekt selbst erfolgen, oder durch Interaktion mit anderen Objekten. Die Interaktion zwischen den Objekten kann *indirekt* durch das Benutzen der Zustände der Attribute von anderen Objekten als Eingangsvariablen für die Transitionsfunktion geschehen, oder *direkt*. Damit ist gemeint, dass ein Objekt ein anderes Objekt bzw. dessen Attribute direkt modifizieren kann, ohne zuerst den eigenen Zustand zu ändern. Als Beispiel kann hier wieder das Objekt vom Typ *Pheromon* mit dem Attribut *Intensität* herangezogen werden. Die Erzeugung eines Pheromons durch einen Agenten, wie oben beschrieben, ist eine Modifikation. Wenn der Agent eine Aktion ausführt, die das Pheromon erzeugt, dann ist das eine direkte Modifikation, ändert er hingegen nur seinen eigenen Zustand, welcher dann von einem Objekt Pheromon gelesen wird und zu seiner eigenen Erzeugung führt, dann handelt es sich um eine indirekte Modifikation. Obwohl es nicht unbedingt der intuitiven Vorstellung von Objekten entspricht, dass nicht existente Objekte die Zustände anderer Objekte lesen können und sich selbst danach erzeugen, so ist es im Modell dennoch möglich, da die Existenz eines Objektes ja nur durch den Zustand codiert ist. Man könnte auch sagen, die Existenz eines Objektes ist eine seiner Eigenschaften. Ein leichter nachvollziehbares Beispiel für indirekte Modifikation ist ein System mit den Objekten *Ampel* und *Autofahrer*. Die Ampel verändert ihren eigenen Zustand (z. B. das Attribut *Signal* von *rot* nach *grün*), dieser wird vom Autofahrer gelesen und bewirkt dort eine Zustandsänderung (z. B. das Attribut *Gemütszustand* von *ungeduldig* nach *erfreut*).

Darüber hinaus kann man zwischen der Modifikation von Objekten auf der Zelle selbst und Objekten auf Nachbarzellen unterscheiden. So könnte ein Agent die Pheromone auf einer Nachbarzelle ablegen (direkte *Nachbar-Modifikation*). Eine indirekte Nachbar-Modifikation findet durch den Einfluss der Zustände der Nachbarzellen auf die Transitionsfunktion statt, z. B. wenn sich die Ampel und der Autofahrer nicht auf der gleichen Zelle befinden.

Zunächst soll der Fall ohne direkte Modifikation betrachtet werden, da er der einfachste ist. Die Attribute der jeweiligen Objekte werden dann nur von den Regeln des eigenen Objekts verändert (Abb. 3.3). Jedes Objekt verfügt über einen Sensor, der aus sämtlichen Eingangsvariablen die nicht verwendeten filtert und nur die relevanten als Wahrnehmung weitergibt. Die Funktionen $\phi_A, \phi_B, \ldots$ beinhalten die Kontrollfunktion und den Aktuator. Außerdem realisieren sie die Wirkung, indem sie direkt den neuen Wert der Attribute ausgeben. Die Aktionen und Entscheidungen sind in dieser Struktur nicht direkt sichtbar. Neben den Attributen aller Objekte wird auch der Zelltyp τ durch die Funktionen $\phi_A, \phi_B, \ldots$ verändert. Bei der Veränderung des Zelltyps sind alle dynamischen Objekte zu berücksichtigen, die aufgrund der Eingangsvariablen gelöscht oder erzeugt werden können. Die Funktion ϕ_τ bestimmt den neuen Zelltyp τ. Diese Funktion ist zusätzlich zu den separaten Funktionen der Objekte notwendig, da alle Objekte und deren Entscheidungen einen Einfluss auf den Zelltyp. Alle Funktionen zusammen (inklusive der Sensoren) ergeben die ursprüngliche Transitionsfunktion ϕ. Die Struktur mit separaten Objekten ist abbildbar auf die allgemeine Struktur aus Abb. 3.1. Die Funktionen $\phi_A, \phi_B, \ldots$ und ϕ_τ haben (entsprechende Sensoren vorausgesetzt) die gleichen Eingangsvariablen wie ϕ und

können somit die gleiche Funktion realisieren, die die Werte für die Attribute und den Zelltyp generiert.

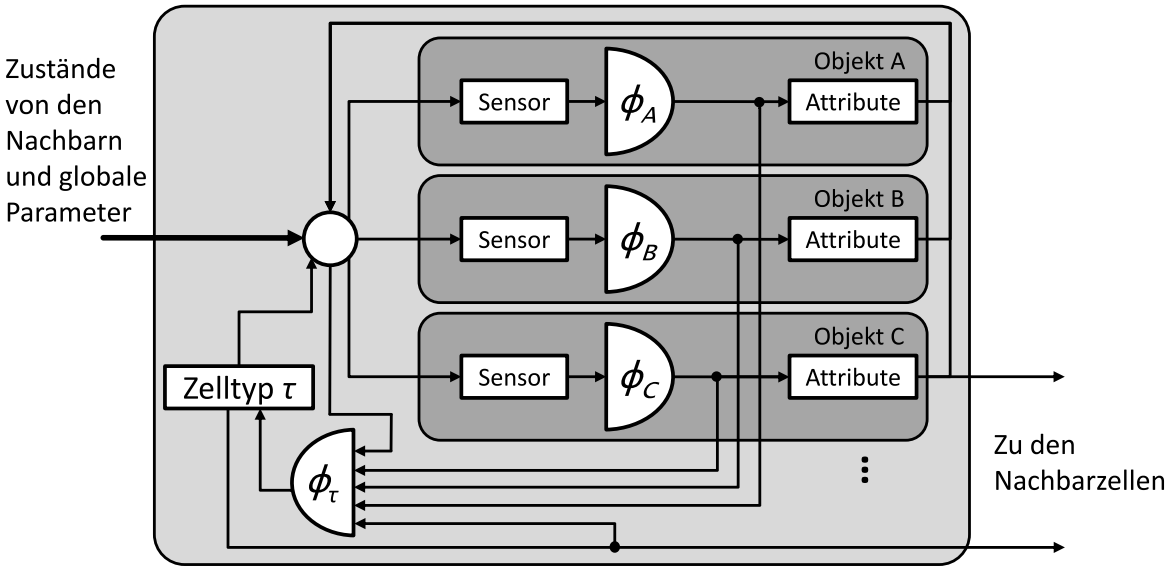

Abb. 3.3.: Struktur der Zellen im MAS mit separaten Transitionsfunktionen. Die Objekte umfassen in dieser Darstellung all ihre Attribute und die Regeln bzw. das Verhalten. Zur besseren Übersichtlichkeit sind hier nur drei Objekte dargestellt.

MAS, in denen alle Objekte, also auch Agenten, lediglich sich selbst direkt modifizieren können, sind ein sehr einschränkender Spezialfall, da nur eine indirekte Kommunikation und Interaktion möglich ist. Die Pheromone z. B. können so nicht modelliert werden, da sie von den Agenten erzeugt, und damit direkt modifiziert werden. Nehmen wir außerdem wie oben wieder an, die Intensität der Pheromone soll sich in jedem Zeitschritt um 1 dekrementieren und dass ein Agent unter bestimmten Bedingungen nicht nur Pheromone erzeugt, sondern auch deren Intensität um 5 erhöhen kann. In diesem Fall tritt zunächst scheinbar ein Konflikt auf, denn die Intensität kann nicht gleichzeitig um 5 erhöht und um 1 verringert werden. Da es sich bei der Intensität um einen Skalar handelt, liegt es Nahe, dass man beide Effekte miteinander verrechnet. Das bedeutet aber, dass die Transitionsfunktion des Agenten, die die Aktion der Erhöhung als Reaktion auf den gegenwärtigen Zustand der Eingangsvariablen bestimmt hat, auf die Attribute des Pheromons Einfluss hat.

Ein allgemeinerer Fall, in dem zwei Objekte gegenseitig modifizierenden Einfluss auf ihre Attribute haben, die keine Skalare sind, ist folgendes simples Beispiel: Gegeben sind zwei Agententypen, *Räuber* und *Beute*. Der Räuber hat zwei mögliche Aktionen, *zuschnappen* und *nicht zuschnappen*, während die Beute die zwei Aktionen *flüchten* und *nicht flüchten* ausführen kann. Der Räuber hat das Attribut *Sättigung*, das entweder im Zustand *satt* oder *hungrig* sein kann. Die Beute hat das Attribut *Gesundheit*, das die Zustände *gesund* und *verletzt* annehmen kann. Angenommen, ein hungriger Räuber und ein Agent des Typs Beute (gesund) sind auf einer Zelle vorhanden, dann gibt es vier mögliche Kombinationen von gleichzeitigen Aktionen. Nur wenn der Räuber zuschnappt und die Beute nicht flieht, soll der Räuber danach satt sein und die Beute verletzt. In den anderen Fällen bleibt die Beute gesund und der Räuber hungrig. Die Attribute beider Agenten sind in diesem Szenario von beiden Aktionen abhängig.

Im allgemeinen Fall könnten also alle Objekte alle anderen Objekte direkt beeinflussen (Abb. 3.4). Aus den selektierten Eingangsvariablen werden in den *Transitionsfunktionen* $\phi_{KA}, \phi_{KB}, \ldots$, die der Kontrollfunktion entsprechen (der Übersicht halber sind in der Abbildung nur drei Objekte enthalten), Entscheidungen e bestimmt. Bei diesen Entscheidungen handelt es sich nicht um eine direkte Veränderung der Werte in den Attributen der anderen Objekte sowie in den eigenen Attributen, sondern um eine Reaktion des Objekts auf den aktuellen Zustand seiner Umwelt. Die Aktuatoren wandeln die Entscheidungen der einzelnen Objekte $e_A, e_B, \ldots$ in eine Aktion $a_A, a_B, \ldots$ um, und können dabei gegebenenfalls weitere Inputs aus den Nachbarzellen benutzen, um Bedingungen zu überprüfen. Als Bindeglied zwischen der Kontrollfunktion und der Umwelt kann der Aktuator so die Entscheidung eines Agenten relativieren oder auf eine geeignete Weise umsetzen. Das ist sinnvoll in Fällen, in denen die Kontrollfunktion nicht implizit alle Regeln des MAS kennt und befolgen kann, oder wenn die Entscheidungen der Kontrollfunktion nur von abstrakter Form sind und bedingt durch weitere äußere Zustände eine bestimmte Aktion auslösen sollen.

Die Aktionen aller Objekte werden durch einen *Aktionsarbiter* auf Konflikte übreprüft (z. B. die sich gegenläufig verhaltenden Änderungen des Pheromons oder die voneinander abhängigen Aktionen des Räuber-Beute-Beispiels). Aktionen werden gegebenenfalls verrechnet, ignoriert oder priorisiert und für jedes Attribut und den Zelltyp eine *Basisoperation* b ausgegeben. Die Basisoperationen $b_x, b_y, \ldots$ und b_τ werden den *Rechenfunktionen* $\phi_x, \phi_y, \ldots$ und ϕ_τ zugeführt, die die Werte der Attribute und des Zelltyps aktualisieren. Eine Rechenfunktion kann dabei eine Basisoperation aus einer begrenzten Menge von Basisoperationen durchführen, beispielsweise in einem Pheromon, in dem das Attribut *Intensität* um 1 inkrementiert oder dekrementiert werden kann, können die beiden Basisoperatioenen *inkrementieren* und *dekrementieren* verarbeitet werden. Die Rechenfunktionen sind so gestaltet, dass sie einzelne Attribute aktualisieren und nicht den gesamten Datenzustand eines Objekts, der aus mehreren Attributen bestehen kann. Deshalb erfüllt der Aktionsarbiter auch bei Zellen mit nur einem einzigen Objekt eine Funktion. Er muss dann die Aktionen in Basisoperationen für jedes einzelne Attribut aufspalten. Die Rechenfunktion und der Speicher für das Attribut als Einheit wird auch *Rechenwerk* genannt.

In dieser Zellstruktur sind die Objekte und deren Attribute getrennt dargestellt, da über den Aktionsarbiter eine gegenseitige Beeinflussung der Attribute erlaubt sein soll. Dennoch kann man immer noch jedem Objekt die zugehörigen Attribute zuordnen und damit eindeutig bestimmen, ob ein bestimmter Teil der Zelle zu einem Objekt gehört oder nicht. Die gesamte Struktur ist abbildbar auf die Struktur aus Abb. 3.1, denn die Kontrollfunktionen erhalten (entsprechende Sensoren vorausgesetzt) die gleichen Inputs wie die Transitionsfunktion ϕ und können damit Entscheidungen generieren, die über den Aktuator und Aktionsarbiter dieselben Veränderungen am Zustand der Zelle bewirken wie eine beliebige Funktion ϕ.

Zellstruktur für direkte Modifikationen auf Nachbarzellen. Direkte Modifikationen anderer Objekte auf der gleichen Zelle sind mit dem bisher entwickelten Modell möglich, direkte Nachbar-Modifikationen aber nicht. Die naheliegendste Möglichkeit das Modell so zu erweitern, dass es solche Modifikationen erlaubt, ist, die Entscheidungen aus den Nachbarzellen abzugreifen. Dabei bewegt man sich aber vom Modell des CA weg, denn die Transitionsfunktion, die von einem diskreten Zeitpunkt t in den nächsten $(t+1)$ führt, kann als Eingangsvariablen nur die aktuellen Zustände und globalen Parameter benutzen. Die Entscheidungen sind ja zum Zeitpunkt t noch nicht daraus berechnet worden. Es muss also eine Möglichkeit geschaffen werden, bei der innerhalb der Transitionsfunktion ϕ sowohl die Entscheidungen der Objekte in der eigenen Zelle, sowie die der Nachbarzellen berechnet werden. Kurzum, es ist notwendig, dass die Senso-

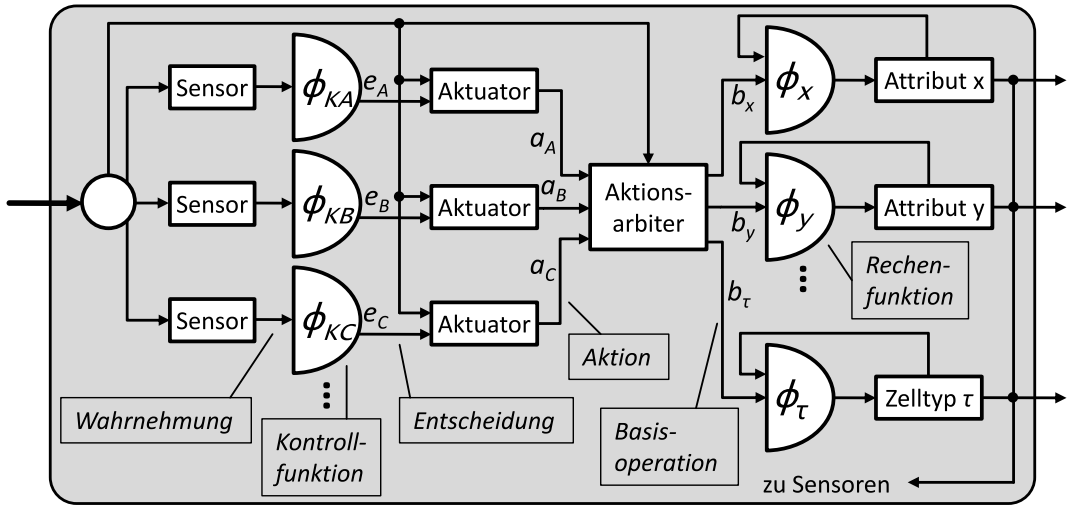

Abb. 3.4.: Struktur einer Zelle im MAS mit gegenseitiger Interaktion zwischen den Objekten. Zur besseren Übersicht sind hier nur drei Objekte dargestellt.

ren, die Kontrollfunktion (die Funktionen $\phi_{KA}, \phi_{KB}, \ldots$) und die Aktuatoren aller Nachbarzellen in der Zelle repliziert sind, damit deren Ergebnisse für den Aktionsarbiter zur Verfügung stehen. Die Gesamtheit aus Sensor, Kontrollfunktion und Aktuator wird als *Kontrolleinheit* definiert.

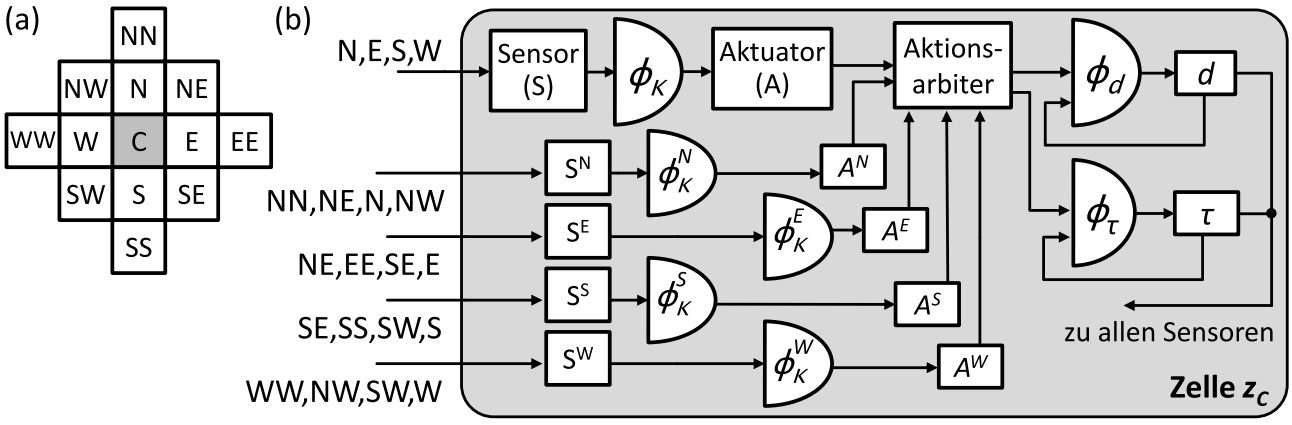

Abb. 3.5.: Struktur einer Zelle in einem MAS mit einem Objekt und replizierten Kontrolleinheiten in der von Neumann-Nachbarschaft. Der Übersicht halber sind hier die Ausgänge zu den anderen Zellen und die Eingänge der Nachbarn zu den Aktuatoren nicht dargestellt. Zur Bezeichnung der Zellen z_i werden nur die Indizes i verwendet.

Abb. 3.5 zeigt eine Zellstruktur für eine Zelle mit einem einzigen Agenten mit einem Attribut d. Die Verwendung von replizierten Funktionen macht allerdings eine Erweiterung der Nachbarschaft notwendig (Abb. 3.5(a)), denn für die Berechnung der Entscheidungen der Nachbarzellen müssen auch die Eingangsvariablen der Nachbarzellen verwendet werden, welche von den Nachbarn der Nachbarn kommen. Das bedeutet eine Erhöhung des Radius um r für Nachbarschaften, die mit einem Radius r definiert sind. Die Nachbarzellen werden nach den Himmelsrichtungen z_N, z_E, z_S, z_W (*North*, *East*, *South*, *West*) genannt, z_C (*Center*) ist die Zelle, auf die sich die Nachbarschaft bezieht, $z_{NN}, z_{NW}, z_{NE}, z_{WW}, z_{EE}, z_{SW}, z_{SE}$ und z_{SS} sind die Bezeichnungen für die weiter entfernten Nachbarn. In MAS, in denen nur ein Teil der Objekte von

Objekten auf Nachbarzellen direkt modifiziert werden kann, muss nur der entsprechende Teil der Kontrolleinheiten repliziert werden.

Abb. 3.6 zeigt beispielhaft den gegenüber Abb. 3.2 erweiterten modifizierbaren Ausschnitt der Umwelt aus Sicht des Agenten (nicht aus Sicht der Zelle wie in Abb. 3.5), der als Dreieck dargestellt ist.

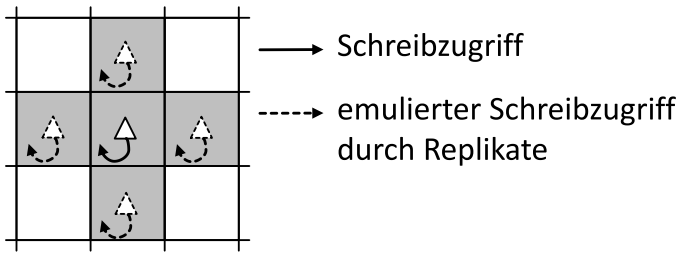

Abb. 3.6.: Modifizierbare Ausschnitte der Umwelt mit replizierten Kontrollfunktionen (gestrichelte Dreiecke) aus Sicht des Agenten in der mittleren Zelle (Dreieck).

Für jedes MAS ergeben sich durch die jeweiligen Eigenschaften der Agenten und passiven Objekte sowie deren Regeln unterschiedliche Möglichkeiten zur Optimierung des Modells hinsichtlich der benötigten Ressourcen in der Zellstruktur. Insofern muss für jedes MAS individuell eine Zellstruktur entwickelt werden, die die allgemeine Struktur konkretisiert. Es können aber allgemeine Ansätze zur Optimierung diskutiert werden, die für typische MAS so oder in ähnlicher Form anwendbar sind.

Eine Idee zur Verringerung der Ressourcen ist, anstelle der Replikation aller Agenten der Nachbarn, nur eine Kontrolleinheit pro Zelle zu verwenden, die auch nur eine Aktion berechnet. Das kann die Aktion eines Agenten sein, der sich gerade auf der Zelle selbst befindet, oder die Aktion eines Agenten auf einer Nachbarzelle. Die Reduzierung der Anzahl der Kontrolleinheiten der Agenten ist besonders erstrebenswert unter der Annahme, dass dem Verhalten der Agenten eine komplexe Modellierung zugrunde liegt, die viele Ressourcen verbraucht. Das Prinzip ist aber nur anwendbar, wenn man einige Einschränkungen macht, d. h. man muss über die Regeln sicherstellen, dass auch immer nur maximal eine Kontrolleinheit für einen Agent diesen Typs benötigt wird, also bei solchen MAS, in denen eine direkte Nachbar-Modifikation nur dann stattfinden kann, wenn sich auf dieser Nachbarzelle kein Agent gleichen Typs befindet. Ein typischer Fall ist die Modellierung der Bewegung eines Agenten (Abschn. 3.1.3). Für die MAS, die innerhalb dieser Einschränkungen modelliert werden können, würde dies jedoch eine Optimierung darstellen.

Wenn nur eine Kontrolleinheit verwendet wird, muss die Zellstruktur angepasst werden (Abb. 3.7). Es müssen die Eingänge aller Nachbarn aus der erweiterten Nachbarschaft im einzigen Sensor vorverarbeitet werden, d. h. der Sensor muss anhand der Eingänge prüfen können, welches der Objekte in der Nachbarschaft, in der Kontrollfunktion berechnet werden soll, und wählt dementsprechend die richtigen Wahrnehmungen aus, um sie an die Kontrollfunktion weiterzugeben. Der Sensor ist damit nicht nur ein Filter der Inputs auf die Kontrollfunktion der jeweiligen Agenten, sondern er übernimmt auch eine Vorverarbeitung der Signale, indem er bestimmte fixe Regeln auswertet. Handelt es sich um eine Kontrollfunktion mit internem Kontrollzustand s, muss dieser auch von der richtigen Nachbarzelle gelesen und verwendet werden. Dazu wird ein Multiplexer eingefügt, der vom Sensor gesteuert wird. Die Kontrollfunktion berechnet dann die Entscheidung e und den neuen Kontrollzustand s'. Der Kontrollzustand, der gespeichert wird,

ist also potentiell der Kontrollzustand eines Agenten, der sich gar nicht in der Zelle befindet. Das ist aber kein Problem, da sich in diesem Fall gar kein Agent in der Zelle befindet (sonst hätte man die Kontrolleinheit ja nicht für einen anderen Agenten benutzen können) und somit auch kein noch benötigter Zustand überschrieben wird.

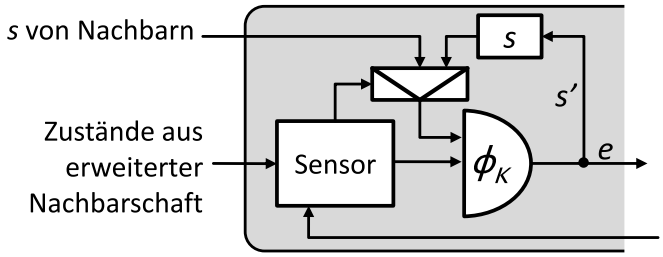

Abb. 3.7.: Erweiterung der Zellstruktur ohne replizierte Kontrolleinheiten. Ein Multiplexer wählt den Zustand eines Agenten aus der Nachbarschaft aus.

Aktionsmapping und Inputreduktion. Bei Agenten, die in der Lage sind, passive Objekte zu manipulieren, z. B. zur indirekten Kommunikation über Pheromone, wird die Aktion a durch den Aktionsarbiter auf eine oder mehrere Basisoperationen abgebildet und zur richtigen Rechenfunktion weitergeleitet, die die Intensität des Pheromons verändert. Ebenso können Aktionen von anderen Agenten eine Wirkung auf die eigenen Attribute haben. Man könnte dies als unbewusste Beeinflussung oder Beeinflussung gegen den „Willen“ des Agenten sehen. Auf diese Weise können auch Regeln des MAS befolgt werden, die der Agent nicht zu kennen braucht. Z. B. muss der Agent nicht wissen, ob er sich tatsächlich bewegen kann, wenn er eine Bewegung als Entscheidung ausgibt. Die Kontrollfunktion könnte eine Entscheidung ausgeben, die eine Bewegung vorsieht, aber der nachgeschaltete Aktuator bekommt einen weiteren Input, der dazu führt, dass die Bewegung nicht stattfindet. Oder man modelliert es umgekehrt so, dass die Kontrollfunktion eine Entscheidung ausgibt, die unabhängig davon ist, ob sich der Agent bewegt oder nicht, und der nachgeschaltete Aktuator „addiert“ abhängig von einem weiteren Input eine Bewegung. Dieser Input kann aus den Zuständen der Nachbarzellen oder der eigenen Zelle kommen, z. B. um Konfliktsituationen zu bestimmen. Ein weiterer Fall so einer unwillkürlichen Beeinflussung kann durch den Aktionsarbiter realisiert werden. Dieser kann nun auch die Aktionen von anderen Objekten als Input verwenden, um die Aktion des Agenten zu manipulieren.

Etwas intuitiver könnte man formulieren: Der Agent wird von anderen Objekten oder von den Regeln des MAS daran gehindert, seine gewünschte Aktion auszuführen. Oder im umgekehrten Fall: Der Agent muss sich mit der Aktion, die der Aktuator bestimmt und der Wirkung, die der Aktionsarbiter (durch Ausgabe von Basisoperationen) bestimmt, abfinden und kann diese nur durch zusätzliche eigene Entscheidungen erweitern/verfeinern. In beiden Fällen entspricht die Menge aller möglichen Entscheidungen (*Entscheidungsmenge*) nicht zwangsläufig der Menge der möglichen Aktionen (*Aktionsmenge*) und auch nicht der Menge von tatsächlich ausführbaren (von außen für andere Agenten oder auch Beobachter des MAS sichtbaren) Wirkungen. Die Entscheidungsmenge kann eine Teilmenge der Aktionsmenge sein, oder die Aktionsmenge selbst enthalten, aber es ist auch möglich, dass die Schnittmenge der beiden leer ist. Die Möglichkeit der Beeinflussung kann auch für eine Form der direkten Kommunikation zwischen den Agenten

genutzt werden. Ein Agent kann dem anderen auf diese Weise eine Information (gegebenenfalls auch „ungewollt“ vom Empfänger) in seine Attribute geben.

Der Aktuator kann ein *Aktionsmapping* durchführen, welches dazu dient, die Größe der Kontrollfunktion über die Anzahl ihrer Outputs klein zu halten. Bei der Optimierung des Agentenverhaltens, also der Optimierung der Kontrollfunktion, kann der Aktuator zur Auslagerung von festen Verhaltensregeln, die nicht weiter optimiert werden sollen, dienen. Beim Aktionsmapping werden von der Kontrollfunktion Entscheidungen e ausgegeben. Der Aktuator bekommt neben den Entscheidungen noch weitere Inputs. Anhand einer Tabelle wird dann aus den Informationen eine Aktion a generiert, welche dann dem Aktionsarbiter als Eingabe dient.

Ein Beispiel für ein Aktionsmapping ist die Auslagerung der Entscheidung des Agenten zur Vorwärtsbewegung. Angenommen ein Agent besitzt eine von vier Himmelsrichtungen und die Aktionen $a \in \{Rs, Ls, Rm, Lm\}$ mit der Bedeutung, sich um 90 Grad nach links (Ls) oder nach rechts (Rs) zu drehen oder sich zu drehen und dabei gleichzeitig auf die Nachbarzelle in momentaner Richtung (die Richtung vor der Drehung) zu bewegen (Rm, Lm). Die Kontrollfunktion könnte lediglich eine Entscheidung $e \in \{R, L\}$ ausgeben. Das Aktionsmapping bekommt diese Entscheidung und ein weiteres Input i (z. B. aus den Nachbarzellen) und generiert daraus die Aktion $a = Xm$ mit $e = X$, falls $i = 0$, sonst $a = Xs$. In diesem Fall nimmt das Aktionsmapping dem Agenten die Entscheidung über seine Vorwärtsbewegung ab und ermöglicht dadurch eine Reduzierung der Outputs. Alternativ könnte der Agent auch eine Entscheidung $e \in \{Rs, Ls, Rm, Lm\}$ generieren. Das Aktionsmapping würde dann je nach Funktion diese Entscheidung wieder revidieren und z. B. aus der Entscheidung $e = Rm$ die Aktion $a = Rs$ machen. Das reduziert zwar keine Outputs der Kontrollfunktion, lagert aber dennoch einen fixen Teil des Entscheidungsmechanismus aus der Kontrollfunktion aus und macht diese somit weniger komplex.

Eine wichtige Funktion kommt auch dem Sensor zu. Um die Kontrollfunktion möglichst kompakt zu halten, muss die Anzahl der Kontrollzustände, Eingänge und Ausgänge gering gehalten werden. Einerseits ist der Sensor dazu da, die relevanten Signale der Nachbarzellen, der eigenen Zelle und eventuell globaler Parameter auszuwählen. Er kann aber auch dazu dienen, eine Vorverarbeitung der Signale durchzuführen und ein kleineres Set von Signalen für die Kontrollfunktion bereitstellen, die bereits Informationen enthalten, die sich nur aus Kombinationen von Eingangssignalen ergeben. Für eine über die reine Selektion hinausgehende Reduktion der Eingangssignale wird der Begriff *Inputreduktion* verwendet.

Ein Beispiel für eine Inputreduktion ist die Vorverarbeitung von der Intensität eines Pheromon-Objektes. Für das Verhalten eines Agenten ist möglicherweise die Intensität nicht so entscheidend, so könnte die Inputreduktion aus der Intensität ein binäres Signal generieren, welches der Kontrollfunktion nur die Information liefert, ob Pheromone mit einer Intensität größer null vorhanden sind oder nicht. Für einen anderen Agenten könnte es z. B. unwichtig sein, in welcher der direkten vier Nachbarzellen sich ein bestimmtes Objekt befindet, sondern nur ob es in einer der Zellen ist. Auch dies kann mit der Inputreduktion auf ein binäres Signal reduziert werden. Allgemein kann man von einem Mapping aller Inputs der Zelle auf eine reduzierte (oder auch erhöhte) Zahl von Inputs für die Kontrollfunktion sprechen. Die Inputreduktion besteht aus einer rein kombinatorischen Logik und kann z. B. als Tabelle modelliert werden.

Abhängig von der Bedeutung der einzelnen Komponenten für die jeweilige Verwendung eines MAS können die Inputreduktion und das Aktionsmapping auch als Teil der Kontrollfunktion interpretiert werden. Dann lägen sie außerhalb des Sensors und des Aktuators.

3.1.3 Bewegliche Objekte

Ein Spezialfall der direkten Nachbar-Modifikation ist der Fall der Bewegung eines Objekts von einer Zelle auf eine Nachbarzelle. Die Bewegung von Agenten wurde bereits in Abschn. 2.3.3 diskutiert und Modellierungsvorschläge aus der Literatur, die Konflikte vermeiden können, wurden besprochen. Hier soll nun ein Bewegungsmodell vorgestellt werden, das in das bisher präsentierte Gesamtmodell passt.

Unabhängig davon, ob sich ein Agent aus seiner eigenen Entscheidung heraus bewegt, oder ob ein passives Objekt bewegt wird, wird eine Bewegung von einer Quellzelle z_1 auf eine Zielzelle z_2 im Modell dadurch realisiert, dass das Objekt auf der Quellzelle gelöscht und auf der Zielzelle erzeugt wird. Das Erzeugen auf der Zielzelle entspricht einer direkten zellübergreifenden Nachbar-Modifikation. Wie diese Modifikationen prinzipiell modelliert werden, wurde bereits beschrieben. Eine Replikation der Kontrolleinheit der Nachbarzellen und die Erweiterung der Nachbarschaft um einen Grad (die Eingangsvariablen der Nachbarn der Nachbarn müssen vorhanden sein) sind nötig. Im Spezialfall der Bewegung kommt hinzu, dass beide Zellen einen konsistenten Übergang berechnen müssen, d. h. wenn das Objekt in der Zielzelle erzeugt wird, dann muss es in der Quellzelle gelöscht werden. Darum müssen beide Zellen zur Berechnung des Folgezustands auch auf dieselben Informationen, sprich Eingangsvariablen, zugreifen können. Das Erzeugen eines Objekts gleichen Typs in der Zielzelle reicht aber nicht aus, um eine Bewegung zu modellieren. Es muss auch eine exakte Kopie des Objektes erstellt werden. Das bedeutet, dass nicht nur der Zelltyp angepasst werden muss, sondern auch die Attribute des Objekts kopiert werden müssen. Eine Veränderung der Attribute von Zeitpunkt t zum Zeitpunkt $t + 1$ gemäß der Regeln des MAS muss dabei auch berücksichtigt werden. Wenn also ein sich bewegendes Objekt ein Attribut x besitzt, dass einerseits in die Zielzelle kopiert werden muss und andererseits durch die Aktion des Objekts (oder auch der eines anderen Objekts) verändert werden kann, muss eine Anpassung des Rechenwerks vorgenommen werden.

Analog zum Vorgehen beim Kopieren eines Kontrollzustands wird ein Multiplexer eingesetzt, der vom Aktionsarbiter kontrolliert wird und den Attributwert der richtigen Nachbarzelle (der Quellzelle) an die Rechenfunktion weiterleitet (Abb. 3.8). Der Aktionsarbiter kennt alle Aktionen und kann daher auch entscheiden, von welchem Agenten die Attribute gegebenenfalls kopiert werden müssen. Falls aufgrund der Regeln des MAS nur eine Kontrolleinheit benötigt wird, kann der Sensor zur Kontrolle des Multiplexers eingesetzt werden, wie für den Kontrollzustand in Abb. 3.7 dargestellt.

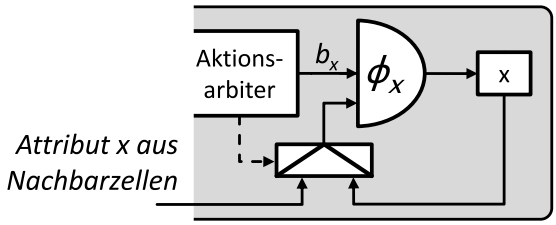

Abb. 3.8.: Erweiterung des Rechenwerks für Attribute von beweglichen Objekten. Ein Multiplexer wählt das Attribut von der Quellzelle zum Kopieren und Modifizieren aus.

Bewegung eines Agenten ohne Kollisionen. Ein konkretes Beispiel anhand der Struktur aus Abb. 3.5 soll das Prinzip verdeutlichen. Angenommen wird ein zweidimensionaler CA und eine von Neumann-Nachbarschaft mit Radius 1 für die Modifikation der Objekte auf der eigenen Zelle, woraus insgesamt für das CA-Modell eine Nachbarschaft mit Radius 2 für das Ermöglichen der direkten Modifikation der Nachbarzellen folgt. Weiterhin wird angenommen, dass in der Zelle (Abb. 3.5(b)) maximal ein einziges Objekt vorhanden sein darf. Der Zelltyp kann dann mit den Zuständen $\tau \in \{leer, voll\}$ codiert werden. Das Objekt selbst ist ein Agent und besitzt ein Attribut namens d, welches ganzzahlige Werte zwischen 0 und 3 annehmen kann, die jeweils auf eine der vier möglichen Bewegungsrichtungen abgebildet werden können. Damit kann bei der Bewegung des Agenten eine Richtung ausgewählt werden. Ferner soll der Agent die Möglichkeit besitzen, sich von einer Quellzelle $z_1 = z_C$ auf eine Zielzelle $z_2 \in \{z_N, z_E, z_S, z_W\}$ zu bewegen. Die Entscheidungsmenge des Agenten umfasst alle möglichen Ausgaben der Kontrollfunktion. In diesem Fall sind das die Entscheidungen $e_{i,k}$ mit der Bedeutung: „gehe nach z_i und setze d auf k“. In diesem einfachen Beispiel bilden die Aktuatoren die Entscheidungen der Agenten auf Aktionen mit der gleichen Bedeutung ab, ohne dass es eine weitere Bedingung gibt. Die Aktionsmenge (alle möglichen Aktionen, die die Aktuatoren ausgeben können) besteht demzufolge aus allen Aktionen $a_{i,k}$ mit der Bedeutung: „gehe nach z_i und setze d auf k“. Für dieses spezielle Beispiel sind die Entscheidungen also genauso aussagekräftig wie die Aktionen, d. h. man kann von der Ausgabe der Kontrollfunktion direkt auf die Basisoperationen abbilden.

Der Zustand d ist dabei dem Agenten zugeordnet und nicht der Zelle, d. h. wenn sich der Agent bewegt, dann muss der Zustand entsprechend auf der neuen Position des Agenten gesetzt werden. Die Entscheidung des Agenten für eine bestimmte Aktion soll von den Zuständen der Nachbarzellen (Radius 1) abhängen, globale Parameter gibt es in diesem Beispiel nicht, das bedeutet in diesem Fall ist ϕ_K eine Abbildung der Zustände der Zellen z_N, z_E, z_S, z_W, z_C auf eine Entscheidung $e_{i,k}$ oder eine „leere“ Entscheidung $e_{\_,\_}$, für den Fall, dass sich gar kein Agent auf der Zelle befunden hat. Die Abbildung ϕ_K^N ist das Replikat der von ϕ_K aus der Zelle z_N, die direkten Nachbarn von z_N sind die Eingangsvariablen für den Sensor S^N, die ein Replikat des Sensors aus der Zelle z_N ist. Analog werden die Kopien für die anderen Nachbarn verwendet. Die Untermenge der Nachbarschaft, auf die die Eingangsvariablen der Sensoren reduziert werden, wird im Folgenden auch *Sichtbereich* genannt, da sie aus Sicht eines Agenten, die zu einem bestimmten Zeitpunkt wahrnehmbare Umwelt und den Agenten selbst enthält. dessen wahrnehmbare Umwelt plus sich selbst darstellen. Der Sichtbereich entspricht hier der von Neumann-Nachbarschaft mit Radius 1. Der Aktionsarbiter kann nun aus allen eingehenden Aktionen auswerten, ob aus einer der vier Nachbarn oder der Zelle selbst, ein Agent und sein neuer Zustand d kopiert werden muss, oder ob ein bereits vorhandener Agent gelöscht werden muss (Zelltyp von $\tau = voll$ nach $\tau = leer$), und diese Basisoperation an die Rechenfunktion weitergeben.

Abb. 3.9 zeigt die Bewegung eines Agenten von einer Quellzelle z_1 nach Süden auf die Nachbar- und Zielzelle z_2. Aus Sicht dieser Zielzelle kommt der Agent also aus Norden. Sei zum Zeitpunkt t folgende Ausgangskonfiguration gegeben:

- Zelle z_1: $\tau = voll$; $d = 2$
- Zelle z_2: $\tau = leer$; $d = 0$
- alle anderen Zellen: wie z_2

Es befindet sich demnach nur ein Objekt, nämlich der Agent, im gesamten System. Das ist ein sehr simples MAS, das jedoch zum Zwecke der Demonstration einer Objektbewegung ausreicht. Das Verhalten des Agenten ist durch die Kontrollfunktion gegeben. Die Eingangsvariablen seien τ und d aus der eigenen Zelle sowie alle Werte τ aus den vier Nachbarzellen $(\tau_N, \tau_E, \tau_S, \tau_W)$. Sei weiterhin das Agentenprogramm wie folgt gegeben:

$$\phi_K = \begin{cases} e_{-,-} & \text{, falls } \tau = leer \\ e_{i,k} & \text{, sonst} \end{cases}$$

wobei $i = C$ und $k = d$, wenn mindestens ein Zelltyp der vier Nachbarzellen *voll* ist. Ansonsten ist $k = d + 7 \mod 4$, $i = N$, falls $d = 0$, $i = E$, falls $d = 1$, $i = S$, falls $d = 2$ und $i = W$, falls $d = 3$. Das bedeutet, der Agent nimmt in seiner Umgebung wahr, ob andere Agenten da sind. Falls ja, bewegt er sich nicht. Falls keine anderen Agenten da sind (in diesem Szenario mit nur einem Agenten immer der Fall), bewegt sich der Agent in eine von d abhängige Richtung auf die Nachbarzelle. Man kann das Attribut daher auch als Codierung einer Bewegungsrichtung des Agenten auffassen. Der Aktionsarbiter gibt abhängig von $a_{i,k}$ bzw. von k, welches vom eigenen Aktuator oder einem der replizierten Aktuatoren kommen kann (falls nicht $a_{-,-}$ geliefert wird), die Basisoperation auf die Änderung von d und den Zelltyp an die Rechenfunktion weiter. Die Rechenfunktionen ϕ_d und ϕ_τ aktualisieren dann den Wert von d bzw. den Zelltyp entsprechend. Dabei muss beachtet werden, dass die Aktionen $a_{i,k}$ von replizierten Aktuatoren ignoriert werden, wenn nicht die entsprechende Richtung (auf die eigene Zelle) darin enthalten ist, d. h. eine Aktion $a_{i,k}$ von A^j wird nur dann für eine Modifikation (modifizierende Wirkung) verwendet, wenn die Richtungen i und j komplementär sind.

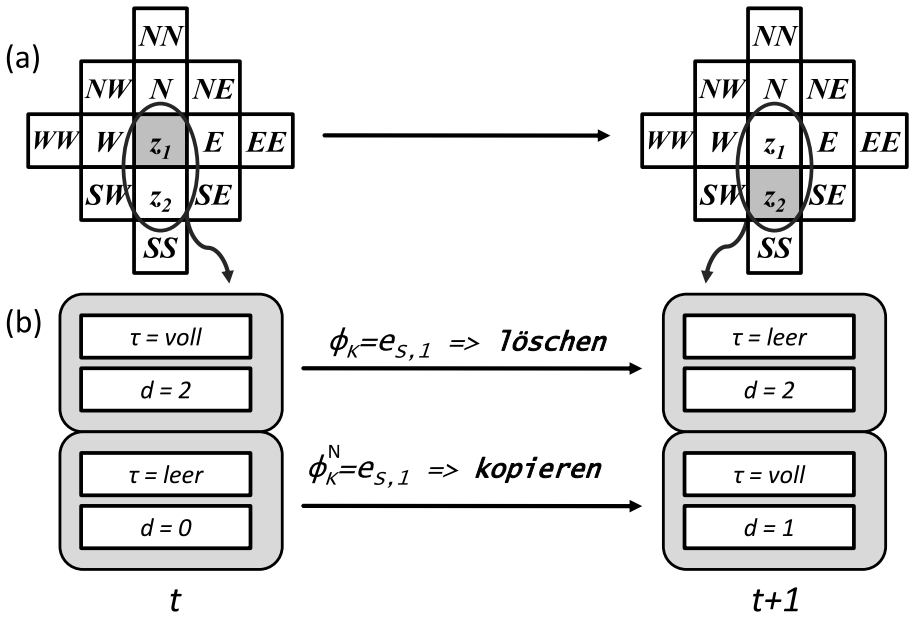

Abb. 3.9.: Beispiel einer Bewegung eines mobilen Agenten von einer Zelle z_1 zur Nachbarzelle z_2. (a) Die markierte Zelle zeigt die Position des Agenten an, die Benennung der Zellen erfolgte im Bezug auf die Quellzelle z_1. (b) Die Veränderung des Zustands in der Quell- und Zielzelle.

Bei der Bewegung sind vor allem die Vorgänge in den Zellen z_1, z_2, z_N, z_E und z_W interessant (Benennung im Bezug auf die Quellzelle, Abb. 3.9(a)). Alle anderen Zellen und deren Nachbarn

mit Radius 1 beinhalten keinen Agenten und daher sind die Entscheidungen aller replizierten Funktionen sowie der Funktion ϕ_K immer „leer“ ($e_{\_,\_}$), was zur Folge hat, dass sich der Zustand dieser Zellen nie ändern kann. Die restlichen Zellen wenden die Regeln wie folgt an:

- Quellzelle z_1: $\phi_K = e_{S,1}$, die replizierten Funktionen generieren die Entscheidung $e_{\_,\_}$. Der Zelltyp wird auf $\tau = leer$ gesetzt, weil die Entscheidung bedeutet, dass der Agent die Zelle in Richtung Süden wechselt. d braucht nicht verändert zu werden, da es nur von Belang ist, wenn sich ein Agent auf der Zelle befindet. Der nächste Agent, der diese Zelle betritt, wird „sein eigenes d mitnehmen“.

- Zielzelle z_2: $\phi_K = e_{\_,\_}$, $\phi_K^N = e_{S,1}$, die anderen replizierten Funktionen generieren die Entscheidung $e_{\_,\_}$. Eine Agentenbewegung aus Norden nach Süden (auf die eigene Zelle) kann von ϕ_d interpretiert werden. Daher wird der Zelltyp auf $\tau = voll$ gesetzt und das Attribut d aus der Reaktion übernommen ($d = 1$).

- Zelle z_N: $\phi_K = e_{\_,\_}$, $\phi_K^S = e_{S,1}$, die anderen replizierten Funktionen generieren die Entscheidung $e_{\_,\_}$. Die Agentenbewegung tangiert die Zelle nicht. Der Aktionsarbiter erhält als Eingangsvariable zwar eine Aktion, die nicht $a_{\_,\_}$ ist, aber deren Ursprungszelle und Bewegungsrichtung nicht zusammenpassen, also generiert ϕ_d auch keine Modifikationen.

- Zelle z_E: $\phi_K = e_{\_,\_}$, $\phi_K^W = e_{S,1}$, die anderen replizierten Funktionen generieren die Entscheidung $e_{\_,\_}$. Die Agentenbewegung tangiert wie bei z_N die Zelle nicht, also generiert ϕ_d auch keine Modifikationen.

- Zelle z_W: $\phi_K = e_{\_,\_}$, $\phi_K^E = e_{S,1}$, die anderen replizierten Funktionen generieren die Entscheidung $e_{\_,\_}$. Die Agentenbewegung tangiert wie bei z_N die Zelle nicht, also generiert ϕ_d auch keine Modifikationen.

Die weiteren Generationen dieses MAS in CA sehen ähnlich aus, nur dass sich die Bewegungsrichtung des Agenten ändert, da sich sein Attribut d verändert. Nach vier Generationen tritt der CA in einen vier Zeitschritte langen Zyklus ein (Abb. 3.10).

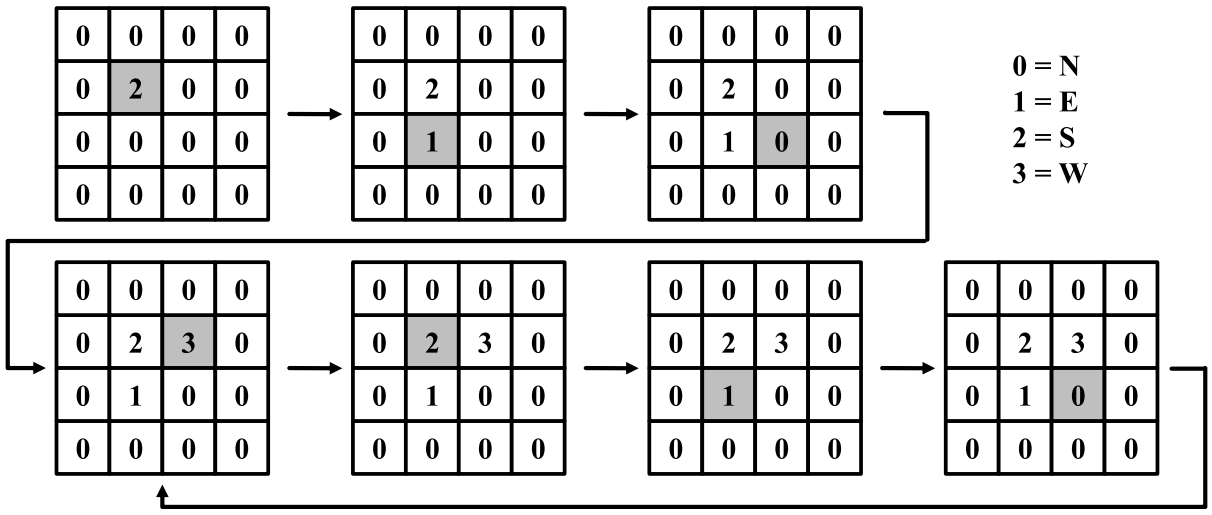

Abb. 3.10.: Sequenz eines MAS mit einem beweglichen Agenten. Die Position des Agenten ist markiert, die Zahl in den Zellen ist der Wert des Atrributs d.

Das Beispiel zeigt zum einen, dass es möglich ist, die autonome Bewegung eines Objekts im CA zu modellieren, es zeigt aber auch, dass dies durch das Replizieren der Kontrolleinheiten viele Ressourcen benötigt. Im Beispiel gibt es auch nur maximal ein einziges Objekt auf jeder Zelle. Will man mehrere sich bewegende Objekte in einer Zelle modellieren, müssten für jedes dieser Objekte die Kontrolleinheiten aus allen Nachbarzellen repliziert werden. Auf das Modell bezogen vergrößert sich dadurch der Zustandsraum (mit dem Zustandsalphabet S) exponentiell mit der Nachbarschaftsgröße auf $|S|^{|N|}$. Hinzu kämen dann auch die Interaktionen zwischen den Objekten, d. h. die Entscheidungen aller Objekte könnten die Rechenfunktionen aller Attribute der anderen Objekte beeinflussen (wie in Abb. 3.4).

Bewegung mehrerer Agenten mit Kollisionen. Bisher wurde die Modellierung der Bewegung anhand eines Beispiels mit nur einem Agenten beschrieben. In diesem Teilabschnitt wird nun untersucht, wie in MAS mit vielen beweglichen Agenten Kollisionen verhindert werden können, so dass die Regeln nicht verletzt werden. Betrachtet werden hier Systeme, in denen Agenten nicht dynamisch sind. *Konflikte* treten auf, wenn die Aktionen der Agenten dazu führen würden, dass sich mehr Agenten als durch die Modellierung der Zellstruktur zulässig auf eine Zelle bewegen wollen und damit eine Kollision stattfinden würde. Die Auflösung dieser Konflikte soll einhergehen mit der Frage, wie der hohe Bedarf an Ressourcen vermindert werden kann.

In Abschn. 2.3.3 wurden bereits einige Strategien zur Konfliktauflösung beschrieben. Dabei wurde meist ein Zeitverlust für ein Kommunikationsprotokoll zwischen Zielzelle und Quellzelle in Kauf genommen (Transactional CA [SFS09] und 2-Phasen-Modell [HHW99]), oder sich auf die Architekturebene begeben (asynchrones Feedback innerhalb eines Taktschritts [HHB06]). Hier soll nun ein Ansatz vorgestellt werden, der es ermöglicht, Konflikte zu lösen, ohne das Modell auf die Architekturebene zu konkretisieren und ohne eine Zeitverzögerung in Kauf zu nehmen. Welche anderen Nachteile stattdessen entstehen, wird im Folgenden erläutert.

Zunächst wird an dieser Stelle ein neuer Typ von Agent eingeführt: Der *gerichtete* Agent. Das ist ein Agent, der eine bestimmte Richtung in den Dimensionen des Zellularen Feldes, in dem er sich befindet, besitzt. Die Richtung ist als Attribut d im Objekt gespeichert und kann sich mit dem Fortschreiten der Generationen ändern, so wie im oben beschriebenen Beispiel das Attribut d. Diese Richtung bezieht auch die Nachbarschaft der Zellen mit ein. In einem zweidimensionalen Feld mit von Neumann-Nachbarschaft ist $d \in \{N, E, S, W\}$, d. h. der Agent „zeigt“ auf eine der Nachbarzellen. Die Richtung d kann durch die Zellregel verändert werden, d. h. auch durch eine Aktion des Agenten selbst. Dem gerichteten Agenten gegenüber steht der *ungerichtete* Agent, welchem das Attribut d fehlt. Dargestellt werden die beiden Agententypen im Folgenden durch Pfeile in eine der Richtungen oder in alle Richtungen (Abb. 3.11). Betrachtet werden MAS, in denen nur jeweils ein Agent auf einer Zelle erlaubt ist. Das vereinfacht die Darstellung und die Konfliktbetrachtung. Konflikte in Systemen, in denen mehr Agenten pro Zelle erlaubt sind, können aber mit dem gleichen Ansatz aufgelöst werden. Sowohl für ungerichtete als auch für gerichtete Agenten wird ein Ansatz vorgestellt.

Konfliktauflösung mit ungerichteten Agenten. Im obigen Beispiel umfasste der Sichtbereich der Agenten die von Neumann-Nachbarschaft mit Radius 1. Der hier vorgestellte Ansatz, die Kollisionen in einem Taktschritt (also ohne mehrstufiges Protokoll) zu verhindern, erfordert ein erweitertes Sichtfeld. Da nur ein Agent pro Zelle erlaubt ist, sind folgende Bewegungen nicht erlaubt:

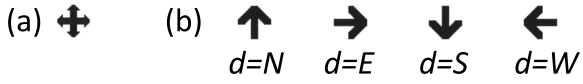

Abb. 3.11.: Darstellungen von (a) ungerichteten und (b) gerichteten Agenten für die von Neumann-Nachbarschaft mit zwei Dimensionen.

- Der Agent kann sich nicht auf eine Zelle bewegen, auf der sich momentan ein Agent befindet, denn er weiß ja nicht, ob dieser sich weiter bewegen wird.
- Der Agent kann sich nicht auf eine Zelle bewegen, auf deren anderen Nachbarzellen sich ein anderer Agent befindet, denn er weiß ja nicht, ob sich dieser andere Agent nicht auch auf die Zelle bewegen will.

Um diese Situationen zu erkennen, muss der Agent einen Sichtbereich mit Radius 2 haben (Abb. 3.12(a)). Mit der Information über alle Konflikte, kann dann eine Bewegung in einer Richtung erfolgen, in der kein Konflikt ist. Zellen, in denen ein Konflikt auftritt werden auch als *Konfliktzellen* bezeichnet.

Leere Zellen (kein Agent im Zelltyp codiert), die einen Agenten in der unmittelbaren Nachbarzelle haben, sind mögliche Zielzellen für die Bewegung des Agenten. Sie brauchen die gleichen Inputs, wie der Agent in der Nachbarschaft, um dessen Reaktion in der eigenen Reaktionsfunktion zu berechnen, die ja sonst nicht benötigt wird, weil kein Agent in der leeren Zelle vorhanden ist. Eine von Neumann-Nachbarschaft mit Radius 3 ist also nötig (Abb. 3.12(b)). Sind mehrere Agenten in unmittelbar angrenzenden Nachbarzellen, so handelt es sich um einen Konflikt und die Zelle braucht keine Entscheidung zu berechnen.

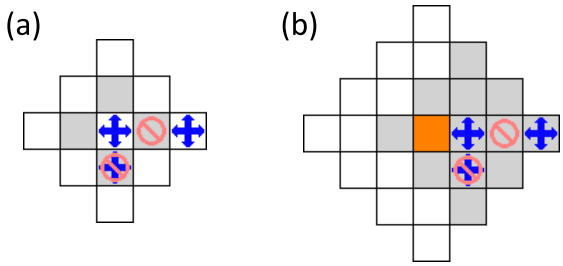

Abb. 3.12.: Konflikte und Sichtbereich von ungerichteten beweglichen Agenten. (a) Sichtbereich des Agenten, mögliche Zielzellen sind hervorgehoben, Konflikte mit einem Verbotszeichen markiert. (b) Die Nachbarschaft der leeren Zelle westlich vom Agenten, hervorgehoben ist der Sichtbereich des Agenten.

Angenommen, es gebe wieder die Zelltypen *leer* und *voll*, wie im obigen Beispiel, dann lässt sich die Zellregel für die Bewegung informal so beschreiben:

- Wenn der Zelltyp *voll* ist, wird überprüft, ob es Konflikte gibt. Dann wird gemäß der Kontrolleinheit eine Bewegungsrichtung ausgewählt („Stehenbleiben“ kann dabei auch eine Option sein). Dabei können die Konflikte mit einbezogen werden oder auch nicht. Der Aktionsarbiter überprüft nun, ob eine Bewegungsrichtung ausgewählt worden ist, die nicht

im Widerspruch zu den Konflikten steht. Dementsprechend wird dann der Zelltyp auf *leer* gesetzt oder er bleibt auf *voll*.

- Wenn der Zelltyp *leer* ist, dann wird überprüft, wie viele Agenten (Zelltyp *voll*) sich in den vier direkt angrenzenden Nachbarzellen befinden. Befindet sich kein Agent dort, oder mehr als einer, bleibt der Zelltyp *leer*. Befindet sich exakt ein Agent dort, dann wird in der eigenen Kontrolleinheit die Aktion des Agenten berechnet. Es findet also dieselbe Berechnung statt, wie in der benachbarten Agentenzelle (Zelle mit Agent). Entsprechend dieser Berechnung wird der Zelltyp auf *voll* gesetzt oder nicht.

Die Bewegungsaktion des Agenten kann noch innerhalb eines diskreten Zeitschritts ausgeführt werden. Allerdings ist hier eine erweiterte Nachbarschaft mit Radius 3 notwendig. Ein weiterer Nachteil ist, dass die Agenten in vielen Fällen einen Konflikt entdecken, und sich deshalb nicht auf jene Nachbarzelle bewegen, obwohl keine Kollision stattfinden würde, weil sich der betreffende andere Agent nicht auf diese Zelle bewegen würde. Diese Entscheidung des anderen Agenten ist aber nicht sicht- oder berechenbar, da ja dafür wiederum die Nachbarschaft um die Nachbarschaft des anderen Agenten erweitert werden müsste. Dessen Entscheidung ist aber wiederum von dessen Konflikten abhängig, so dass sich die Erweiterung letztendlich auf das gesamte Zellulare Feld beziehen müsste.

Konfliktauflösung mit gerichteten Agenten. In Anlehnung an das 2-Phasen-Modell in [HHW99] kann eine Strategie zur Konfliktauflösung entwickelt werden, die mit einer kleineren Nachbarschaft als der von Neumann-Nachbarschaft mit Radius 3 auskommt. Allerdings soll hier im Gegensatz zum 2-Phasen-Modell die Auflösung innerhalb nur eines Taktschritts erfolgen. Dafür können gerichtete Agenten eingesetzt werden.

Der Wert des Attributs d gibt in gerichteten Agenten zu einem bestimmten Zeitpunkt an, auf welche Nachbarzelle sie sich bewegen können. Er ist also schon einen Zeitschritt zuvor berechnet worden. Dies entspricht der Vorgehensweise in [HHW99]. Das Attribut d kann als *Bewegungsrichtung* aufgefasst werden. Der Agent darf sich auf eine freie Zelle in Bewegungsrichtung bewegen, wenn dort kein Konflikt entsteht. Mehrere Fälle von Konflikten können auftreten: Wenn sich auf der Nachbarzelle in Bewegungsrichtung (*Frontzelle*) ein Agent befindet (Abb. 3.13(a)) oder wenn es weitere Agenten gibt, deren Nachbarzelle in Bewegungsrichtung mit der des Agenten übereinstimmt (Abb. 3.13(b)).

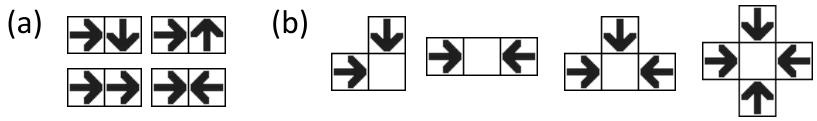

Abb. 3.13.: Konfliktsituationen bei beweglichen gerichteten Agenten. (a) In der Frontzelle befindet sich ein anderer Agent (in der Darstellung aus Sicht des linken Agenten). (b) Mehrere Agenten haben die gleiche Frontzelle.

Der Agent braucht, um alle diese Situationen zu erkennen, einen Sichtbereich innerhalb der von Neumann-Nachbarschaft mit dem Radius 2. Allerdings gehört zu diesem Sichtbereich nur der Teil der Nachbarschaft, der in Bewegungsrichtung liegt. Man kann also von einer Bewegungs- und *Blickrichtung* sprechen (Abb. 3.14(a)). Durch die Blickrichtung entsteht so etwas wie ein dynamischer Sichtbereich, der von der Blickrichtung abhängt. Da es im Modell jedoch keine dynamischen Nachbarschaften gibt, müssen dennoch alle Zellen im Radius 2 in die Nachbarschaft mit einbezogen werden. Der Agent kann mit den Informationen aus

seinem Sichtbereich Konflikte erkennen und dementsprechend eine Bewegung in die Zelle in Bewegungs- und Blickrichtung vollziehen oder nicht.

Die Frontzelle ist die einzig mögliche Zielzelle des Agenten. Sie braucht die gleichen Inputs wie der Agent, um dessen Aktion in der eigenen Kontrolleinheit zu berechnen. Sie steht dafür zur Verfügung, weil sie nur anderweitig benötigt wird, wenn in der Zelle ein Agent vorhanden ist. Hier reicht aber eine von Neumann-Nachbarschaft mit Radius 1 aus (Abb. 3.14(b)). Die Zelle kann anhand ihrer vier direkt angrenzenden Nachbarn feststellen, ob es sich bei ihrer eigenen Position um einen Konflikt handelt. In diesem Fall muss gar keine Aktion berechnet werden.

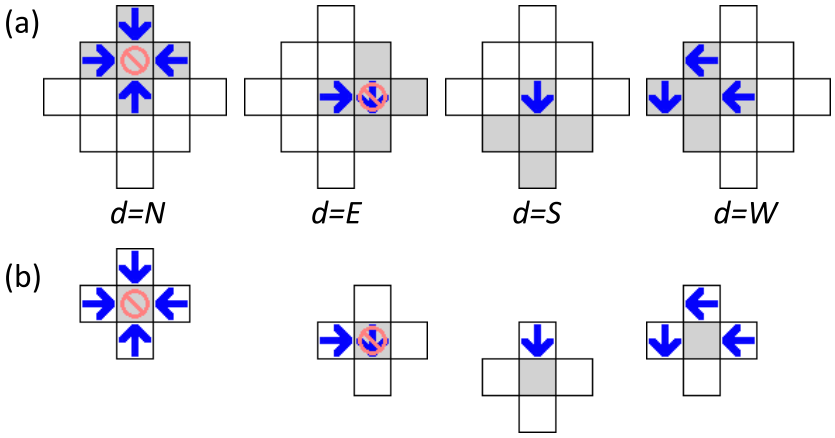

Abb. 3.14.: Sichtbereich und Nachbarschaft von gerichteten beweglichen Agenten. (a) Beispiele für die Nachbarschaft der Agenten für jede Mögliche Blickrichtung. Markiert ist der Sichtbereich des Agenten. Konflikte sind mit einem Verbotszeichen markiert. (b) Die zu (a) gehörigen für die Zielzellen (markiert) relevanten Teile der Nachbarschaft.

Wieder wird angenommen, es gebe die Zelltypen *leer* und *voll*. Informal lässt sich dann die Zellregel für die Bewegung von gerichteten Agenten mit dem Attribut d so beschreiben:

- Wenn der Zelltyp *voll* ist, wird überprüft, ob es in Bewegungsrichtung einen Konflikt gibt. In der Kontrolleinheit wird eine Aktion berechnet, die die Information über den Konflikt mit einbeziehen kann, aber nicht muss. Der Aktionsarbiter berechnet aus der Aktion und der Information über den Konflikt, ob eine Bewegung stattfinden kann oder nicht. Alternativ kann diese Überprüfung auch vom Aktuator vorgenommen werden, wenn eine gegenseitige Beeinflussung von Objekten auf der gleichen Zelle ausgeschlossen ist. Besteht ein Konflikt oder entscheidet der Agent sich für die Aktion „Stehenbleiben“ dann bleibt der Zelltyp auf *voll* und das Attribut d wird entsprechend der Aktion aktualisiert, sonst wird er auf *leer* gesetzt.

- Wenn der Zelltyp *leer* ist, wird überprüft, wie viele Agenten sich in den Nachbarzellen befinden, deren Bewegungs- und Blickrichtung zur Zelle selbst zeigt. Befindet sich dort kein Agent oder mehr als ein Agent mit dieser Bedingung, dann bleibt der Zelltyp *leer*. Befindet sich exakt ein Agent mit dieser Bedingung dort, so wird in der eigenen Kontrolleinheit die Aktion des Agenten berechnet. Entsprechend der Aktion wird der Zelltyp verändert und gegebenenfalls das Attribut d gesetzt.

Durch die erweiterte Nachbarschaft mit Radius 2, die für den sich dynamisch verändernden Sichtbereich der Agenten gebraucht wird, kann das 2-Phasen-Modell in ein Modell mit nur ei-

ner Phase umgewandelt werden, d. h. die Bewegung wird innerhalb eines Taktschritts komplett durchgeführt.

Auch bei der Bewegung der gerichteten Agenten tritt das Problem auf, dass Konflikte die Bewegung von mehreren Agenten verhindern, obwohl sich zumindest einer der betreffenden Agenten bewegen könnte. Dies betrifft alle Fälle aus Abb. 3.13(b). Eine Idee zur Verbesserung dieses Umstands ist die Einführung von *Prioritäten*. Diese sollen bei jedem Konflikt auf einer freien Zelle ähnlich einer Verkehrsampel einem der beteiligten Agenten die Priorität zur Bewegung einräumen. Dazu wird auf jeder Zelle ein nicht bewegliches Objekt *Bewegungsarbiter* mit dem Attribut *Priorität* positioniert. Bei vier Nachbarn ergeben sich 24 Möglichkeiten von Priorisierungsreihenfolgen, z. B. würde bei *Priorität* $= (N, E, S, W)$ der Agent aus der nördlich angrenzenden Nachbarzelle zuerst berücksichtigt werden. Wenn dort kein Agent ist, dann wird der Agent aus der östlich angrenzenden Nachbarzelle priorisiert usw. Daraus ergibt sich, dass die Priorität einen von 24 Werten annehmen kann. Der Wert der Priorität kann dabei durch die Regel für den Bewegungsarbiter generationsweise verändert werden oder er bleibt fix. Für das grundlegende Verfahren spielt das keine Rolle. Wichtig ist, dass alle Agenten um die Zielzelle herum aus der Priorität und der Konfliktsituation lesen können, ob ihnen die Bewegung gestattet wird oder nicht. Auf der anderen Seite muss die Zielzelle mit ihrer eigenen Priorität feststellen, welchen Agenten sie kopiert bzw. wessen Aktion sie berechnet (Abb. 3.15).

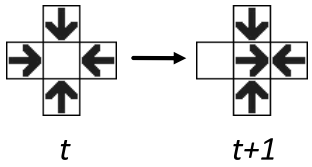

Abb. 3.15.: Beispiel für die Auflösung eines Konflikts mit Prioritäten. Der Bewegungsarbiter der Konfliktzelle hat die Priorität $(W, \ldots)$, daher wird zuerst dem Agent aus westlicher Richtung die Bewegung erlaubt. Die Veränderung der Richtung eines Agenten wäre möglich, ist hier aber nicht dargestellt.

Bei ungerichteten Agenten können solche Bewegungsarbiter nur mit hohem Aufwand eingesetzt werden, da die Konfliktzelle dann für jeden der Agenten in der direkten Nachbarschaft dessen gewünschte Bewegungsrichtung bestimmen müsste, um dann denjenigen zu kopieren, der priorisiert ist und dessen Bewegungsrichtung auf die Konfliktzelle zeigt. Das würde wieder mehrere Replikate der Kontrolleinheit erfordern.

Um noch eine weitere Art des Konfliktes bei gerichteten Agenten auflösen zu können, kann ein Tausch der Positionen vorgenommen werden. In Situationen, in denen sich zwei Agenten gegenüber stehen, also in zwei benachbarten Zellen mit Blickrichtung auf die jeweils andere Zelle (Abb. 3.13(a) rechts unten), kann man solch einen Positionstausch (*Swap*) erzwingen (Abb. 3.16). Da es nicht erlaubt wäre, wenn sich nur einer der beiden Agenten bewegt, hat ein Agent in dieser Situation dann nicht mehr die Möglichkeit, auf der Zelle stehenzubleiben. Es sei denn, beide Agenten entscheiden sich zum stehenbleiben. Dann bräuchte man allerdings wieder ein Replikat der Kontrolleinheit, um die Aktionen beider Agenten gleichzeitig zu berechnen. Aus Sicht der Zelle, muss die Regel so ergänzt werden: Wenn der Zelltyp *voll* ist und der Zelltyp in Blickrichtung ebenfalls *voll* und deren Blickrichtung entgegengesetzt ist, dann wird der Agent aus der Nachbarzelle kopiert.

Inwieweit ungerichtete oder gerichtete Agenten mit Prioritäten und/oder Swap eingesetzt werden können, hängt auch von der Anwendungsform des MAS ab. Bei den problemlösenden

Abb. 3.16.: Beispiel für den Positionstausch von gerichteten Agenten. Die Agenten müssen ihre Positionen tauschen, wenn sie in eine solche Konfliktsituation geraten. Die Veränderung der Richtung eines Agenten wäre möglich, ist hier aber nicht dargestellt. Zur Unterscheidung sind die beiden Agenten unterschiedlich gefärbt.

MAS, die in dieser Arbeit untersucht werden, wird das Verhalten und gegebenenfalls die Interaktionsmechanismen gesucht, um ein gegebenes globales Problem zu lösen. Es sollte also diejenige Variante gewählt werden, die das Problem besser löst. Die Entscheidung für eine dieser Alternativen kann dann auch als Teil des Optimierungsprozesses angesehen werden. Andere Ausprägungen und Arten von Bewegungen sind dabei auch denkbar.

3.1.4 Vollständige Beschreibung eines MAS

Ein MAS kann mit dem entwickelten Modell konstruiert werden, indem die folgenden Elemente spezifiziert werden:

- die Größe und Dimension der Agentenwelt
- die Nachbarschaftsbeziehungen der Zellen
- alle Objekttypen und deren Attribute
- die Anzahl und initialen Positionen und initialen Zustände aller Objekte
- die Regeln für die Objekte, also die Funktionen der Sensoren, der Kontrollfunktion und der Aktuatoren, sowie des Aktionsarbiters

Für MAS, die die Funktionen der einzelnen Komponenten, wie Sensoren, Aktuatoren und Aktionsarbiter, nicht oder nur eingeschränkt verwenden, können diese auch weggelassen oder durch simplere Elemente ersetzt werden. Das in diesem Kapitel entwickelte Modell ist als allgemeines Modell zu verstehen, das man spezifisch für die jeweiligen Bedürfnisse vereinfachen oder auch anpassen kann, um Ressourcen zu sparen oder eine einfachere Beschreibung zu ermöglichen.

3.2 Möglichkeiten zur Modellierung des Agentenverhaltens

Die Kontrollfunktion ist bisher als frei konfigurierbare Blackbox angesehen worden, die Eingabevariablen verarbeitet und daraus Entscheidungen generiert. In diesem Abschnitt sollen Möglichkeiten beschrieben werden, wie diese Blackbox strukturiert sein kann. Da in dieser Struktur die Entscheidungen des Agenten getroffen werden, ist sie letztlich der Hauptauslöser des gesamten Verhaltens. Alle Aktionen, die ein Agent im MAS durchführen kann, jegliche Manipulation seiner Umwelt, seine Bewegung, und das Bewegen anderer Objekte müssen von der Struktur der Kontrollfunktion bereitgestellt werden, sei es auch in abstrakter Form, die der Aktuator zu übersetzen vermag. Die Kontrollfunktion muss sich also in die Kontrolleinheit mit den richtigen Schnittstellen einfügen. Bei Russell und Norvig [RN04, S. 70] wird der Teil des Agenten, der

die vom Agentenprogramm ausgewählten Aktionen durchführt und die Umwelt wahrnimmt, also die Sensoren und Aktuatoren, als *Agentenarchitektur* verstanden. In dem hier entwickelten Modell stellen die Sensoren, Aktuatoren und der Aktionsarbiter die Architektur eines Agenten dar. Durch die Architektur wird der Agent auf bestimmte durchführbare Aktionen beschränkt. Das bedeutet, der Agent kann sich bei seinen Aktionen nicht über die „Naturgesetze“ des MAS hinwegsetzen, z. B. muss die Maximalanzahl von Agenten auf einer Zelle eingehalten werden.

Sowohl Agentenarchitektur als auch Kontrollfunktion sind ein Teil der Zellstruktur und je nach Interpretation kann man das Verhalten lediglich im Agentenprogramm oder zum Teil auch in der Agentenarchitektur begründet sehen. In diesem Abschnitt werden Modellierungsmöglichkeiten für das Verhalten von Agenten im Sinne von *Kontrollfunktion* vorgestellt. Die Menge der Aktionen der Agenten und die spezifischen Manipulationen der Umwelt können im allgemeinen Modell natürlich nicht berücksichtigt werden, sondern müssen beim Design eines konkreten MAS in die Zellregeln eingearbeitet werden.

Das Modell, das in Abschn. 3.1 entwickelt wurde, kann mit verschiedenen Formen der Kontrollfunktion verwendet werden. In diesem und im nächsten Teilabschnitt werden einige dieser Konzepte vorgestellt. Der Schwerpunkt liegt dabei auf *endlichen Zustandsautomaten* (FSM, engl.: Finite State Machine), weil dieses Konzept später in dieser Arbeit für die Untersuchung von Optimierungsverfahren benutzt wird.

3.2.1 Modellierung der Kontrollfunktion mit endlichen Zustandsautomaten

Bei der Wahl eines Mechanismus für die Realisierung eines Agentenprogramms stehen in dieser Arbeit zwei wesentliche Punkte im Vordergrund. Erstens soll das Agentenprogramm möglichst simpel sein. Mit simpel ist hier gemeint, dass die Ressourcen, die für dessen Implementierung benötigt werden, möglichst gering sind. Die Ressourcen gering zu halten ist eine Voraussetzung für die Erhaltung der Skalierbarkeit eines massivparallelen Systems wie dem entwickelten CA-Modell. Außerdem soll es für eine (vollparallele) Implementierung in Hardware (zur Ausnutzung der Parallelität) mit begrenzten Ressourcen geeignet sein. Der zweite Punkt zielt auf die Möglichkeit zur Optimierung des Agentenverhaltens ab. Der Mechanismus der Kontrollfunktion sollte so gestaltet sein, dass es möglich ist, ein gutes Verhalten, welches möglichst effizient zur Problemlösung führt, zu programmieren. Er sollte also also trotz simpler Struktur ein komplexes Verhalten ermöglichen. Dieses Ziel ist nicht trivial, aber möglich. Braitenberg beschreibt beispielsweise in [Bra93] wie man mit scheinbar einfachen Mitteln ein sehr komplexes Verhalten eines „Vehikels“ erzeugen kann. Braitenbergs Vehikel besitzen nur eine einfache Sensorik und einfache Reaktionsregeln. Er argumentiert, dass von außen betrachtet ein komplexes System dahinter vermutet werden muss, während in Wahrheit das ganze Verhalten auf miteinander agierende simple Mechanismen beruht. Der Mechanismus soll außerdem für die Anwendung von (heuristischen) Optimierungsmethoden geeignet sein, um die Problemlösung zu erreichen bzw. die Effizienz zu erhöhen.

Endliche Zustandsautomaten (FSMs) erfüllen die erste Voraussetzung, wenn man die Anzahl der Zustände, Eingänge und Ausgänge beschränkt. In [Hal08] wurden bereits MAS in CA, die FSMs als Agentenprogramm benutzen, untersucht und festgestellt, dass FSMs mit relativ wenigen Zuständen (bis zu 6 Zustände) komplexe Verhaltensweisen ermöglichen. Dort wurden bereits Techniken zur Verhaltensoptimierung von MAS in CA beschrieben. Diese Techniken beruhen im Wesentlichen auf der optimierten Aufzählung aller Möglichkeiten einer durch

Zustandsautomaten kodierten Verhaltensregel und der Beschleunigung dieser optimierten Aufzählung durch geeignete Hardwarearchitekturen. Bei steigender Komplexität des Verhaltens der Agenten stößt der Ansatz allerdings schnell an seine Grenzen. Heuristische Verfahren wurden nicht eingesetzt. Die Anwendung von heuristischen Optimierungsmethoden ist Gegenstand der Untersuchungen in Kap. 5.

In weiteren Arbeiten wurden FSMs zur Modellierung des Verhaltens von Agenten eingesetzt. Beispielhaft seien einige genannt: In [MSPP02], [JCC+90] und [SG00b] wurden FSMs mit beschränkter Anzahl von Zuständen als Agentenprogramme für lernende Agenten eingesetzt und mit heuristischen Verfahren evolviert. Die Aufgabe der Agenten war es in den ersten beiden Quellen, auf einem zweidimensionalen Zellulären Feld, bestimmte Pfade zu beschreiten und in der dritten Quelle, gegen einen Gegenspieler ein Gebiet zu erobern. Komann benutzt in [Kom10] manuell entwickelte und evolvierte Zustandsautomaten für Agenten in CA, deren Aufgabe es ist, Daten aus Bildern von Objekten zu extrahieren. Diese Agenten werden auch in Hardware implementiert. Agentenverhalten für *Search Agent Software* wird in [AAAB09] mit FSMs modelliert. Die Agenten durchsuchen dort Datenbanken nach Buchtiteln und Preisen. Agentenprogramme als FSMs mit bis zu 25 Zuständen werden in [LK05] für animierte virtuelle Charaktere verwendet. Diese Agenten sind in der Lage, sich in dynamisch veränderlichen Umgebungen mit Hindernissen zu bewegen. In [Mar99] wird eine Infrastruktur für problemlösende Umgebungen mit Software-Agenten vorgestellt, die mit FSMs auf mehreren Ebenen modelliert werden. Intelligente Agenten werden auch in [AHY+10] mit FSMs modelliert mit dem Zweck, ein Tool zur intuitiven und schnellen Erstellung von Trainingssimulationen der echten Welt (z. B. für Rettungskräfte) zu schaffen.

Im Bereich der Computerspiele ist die Verwendung von FSMs als Agentenprogramm eine beliebte Methode, um intelligente bzw. intelligent erscheinende Agenten zu produzieren. Buckland beschreibt in [Buc05, S. 43] die Motivation für die Verwendung von FSMs für Computerspielagenten. Dabei nennt er folgende Punkte:

- die Einfachheit, solche zu entwerfen und zu debuggen, da jeder Zustand vom Programmierer separat nachverfolgt werden kann,
- den geringen Berechnungsaufwand, weil die Interpretation einer FSM durch Tabellenzugriff, *if-then-* oder *case*-Anweisungen realisiert werden kann und keine zusätzlichen rechenintensiven Prozesse erfordert,
- die Intuitivität, da die FSMs durch ihre Zustände der Sichtweise von Menschen auf andere Menschen oder auch Objekte entspricht, die alles in einem bestimmten Zustand befindlich ansieht,
- die Flexibilität, weil man die FSM einfach anpassen kann, um das Verhalten eines Agenten zu verändern. Die Anpassung kann durch eine Erweiterung oder Reduzierung der Zustände geschehen oder durch die Veränderung von Zustandsübergängen.

Eine Besonderheit bei Agenten in Computerspielen ist die Zielsetzung beim Entwickeln ihres Verhaltens. Es sind nicht unbedingt Agenten gesucht, die ein Problem optimal lösen können. Die Agenten werden als Gegner oder auch als Mitspieler für den menschlichen Spieler eingesetzt und im Vordergrund steht der Spielspaß. Daher sind oft Agenten mit beschränkter Intelligenz gefordert, die dem menschlichen Spieler eine Chance lassen, das Spiel zu gewinnen, aber gleichzeitig auch eine gewisse Schwierigkeit darstellen.

Ein weiterer Grund für die Verwendung von FSMs als Kontrollfunktion ist die einfache inkrementelle Konstruktion von Agentenverhalten. In [Pet06] wird argumentiert, dass das Hinzufügen von neuen Fähigkeiten durch das Hinzufügen neuer Zustände und Transitionen geschehen kann und relativ wenige Veränderungen an einer bisherigen kleineren FSM erfordert. Dies stehe im Gegensatz zur Notwendigkeit von dramatischen Veränderungen an z. B. neuronalen Netzen.

Durch die Verwendung einer FSM als Agentenprogramm bekommt der Agent einen internen Kontrollzustand. Dieser Zustand ist gemäß dem CA-Modell aus Abschn. 3.1 kein Attribut des Agenten, denn er ist kein Teil des Datenzustands. Er kann aber als Attribut modelliert werden, wenn die Aktionen so angepasst werden, dass eine Zustandsänderung über eine Aktion geschieht. Der dann über eine Rechenfunktion veränderte Kontrollzustand kann über die Sensoren eingelesen und in der Kontrollfunktion benutzt werden. Unabhängig von der semantischen Interpretation dieses Zustands, d. h. die Annahme darüber, welches Wissen der Zustand repräsentiert, kann man *FSM-kontrollierte* Agenten zu den modellbasierten Reflex-Agenten bzw. den hysteretischen Agenten zuordnen. Eine Zielvorgabe oder Nutzenfunktion innerhalb des Agentenprogramms existiert nicht.

In den meisten der erwähnten Beispiele für FSM-kontrollierte Agenten sind die Zustände der FSM der einzige bestimmende Faktor für die Aktionen, die die Ausgabe der FSM darstellen, d. h. es handelt sich um *Moore-FSMs* [Moo56]. Typische Zustände in solchen FSMs für Agenten sind z. B. „fliehen“, „angreifen“, „essen“, „warten“, „springen“ etc. Die Art des Zustands hängt natürlich auch von der Komplexität der Aktionen ab, die von MAS zu MAS variieren kann. Die Aktionen werden dann vom Agenten so lange wiederholt, bis sich der Zustand der FSM ändert. Die FSMs besitzen dabei meist keinen Terminalzustand und induzieren damit als Agentenprogramm ein Verhalten unbestimmter Länge. Dies ist notwendig für den Einsatz in MAS, da die Agenten darin kontinuierlich agieren sollen.

In dieser Arbeit werden *Mealy-FSMs* [Mea55] verwendet, die die Aktionen nicht durch die Zustände, sondern über die Kombination von Inputs und Zuständen definiert. Das hat den Vorteil, dass die Agenten in dem CA-Modell, das ja auf diskreten Zeitschritten (Generationen) beruht, schneller auf die Umwelteinflüsse reagieren können. Bei einem Moore-Automaten muss die aktuelle Aktion schon im vorausgegangenen Takt berechnet worden sein, denn die Ausgabe hängt ja ausschließlich vom Zustand ab. Bei einem Mealy-Automaten kann die Aktion direkt aus den Eingaben und dem aktuellen Zustand bestimmt werden und die aktuellen Attribute der Objekte in der Zelle noch im selben Zeitschritt modifizieren. Hierbei ist zu beachten, dass die Mealy-FSM, die die Aktionen generiert nur der Kontrollfunktion entspricht. In ihrer Gesamtheit, also die Rechenfunktionen der Zelle mit eingeschlossen, stellt die Zelle wieder eine Moore-FSM dar.

Abb. 3.17 zeigt die Zellstruktur für eine Zelle, die einen gerichteten Agenten (mit Richtungsattribut d) aufnehmen kann. Da es sich um bewegliche Agenten handelt, muss die Kontrolleinheit für jeden Nachbarn repliziert in der Struktur enthalten sein. Zur besseren Übersicht ist dies hier nicht dargestellt und der Aktuator und Aktionsarbiter zusammengefasst. Die Kontrollfunktion ϕ_K bekommt als Eingaben den Zustand s und die Eingaben der Nachbarzellen über den Sensor und realisiert damit eine Mealy-FSM. Diese FSM, die das Verhalten des Agenten bestimmt, wird auch *Kontroll-FSM* genannt. Die Ausgaben der Funktion entsprechen dem Folgezustand s', und den Emtscheidungen a des Agenten, welche über den Aktuator und Aktionsarbiter in Aktionen und schließlich Basisoperationen umgesetzt als Eingabe in die Rechenfunktion ϕ_d und ϕ_τ gehen. Die Rechenfunktion ϕ_d bekommt als Eingabe die Basisoperation b_d und den aktuellen Wert d und berechnet daraus den Folgewert d'. Die Rechenfunktion ϕ_τ bekommt als Eingabe die Basisoperation b_τ und den aktuellen Wert τ und berechnet daraus den Folgewert τ'. Es han-

delt sich bei den Rechenwerken um zwei Moore-Automaten. Die hintereinandergeschalteten Automaten kann man auch als Mikroprogramm-Steuerwerk (Kontroll-FSM-Teil) und Operationswerk (Rest) interpretieren, da solche Steuerwerke in ihren einfachen Realisierungsformen FSMs entsprechen [Hof93, S. 197ff.].

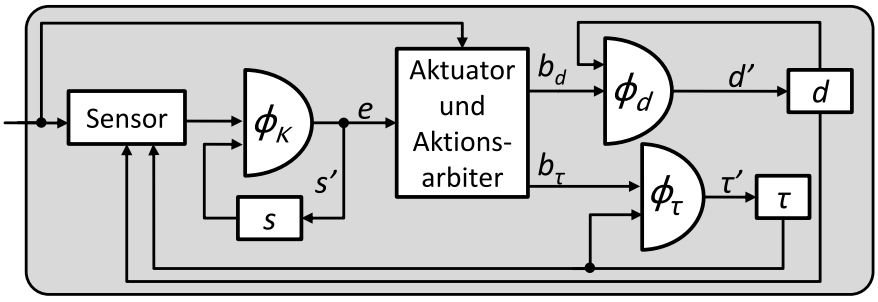

Abb. 3.17.: Struktur einer Zelle mit Mealy-FSM-kontrolliertem Agenten. Der Übersicht halber sind weitere Objekte und die replizierten Kontrolleinheiten weggelassen.

Etwas detailliertere Darstellungsformen der Mealy-FSM sollen anhand eines konkreten Automaten als Beispiel gegeben werden. Er soll in die Zellstruktur aus Abb. 3.17 passen, d. h. es soll eine Mealy-FSM für einen gerichteten Agenten mit Bewegungsmöglichkeit sein. Sei die Menge der Zustände der Kontroll-FSM $s \in \{0,1,2,3,4,5\}$, die Menge der möglichen Inputs der Kontroll-FSM $x \in \{x_0, x_1\}$, die Menge der möglichen Entscheidungen, also Outputs der Kontroll-FSM $e \in \{e_0, e_1\}$, sowie $d \in \{0,1,2,3\}$, $b_d \in \{0,1\}$, $b_\tau \in \{0,1\}$ und $\tau \in \{leer, voll\}$. Der Aktuator bilde eine Entscheidung e_i auf eine Aktion a_i ab. Die einzelnen Funktionen seien wie folgt definiert:

- Aktionsarbiter: $b_d = 1$ und $b_\tau = 0$ für $a = a_0$, $b_d = 0$ und $b_\tau = 1$ für $a = a_1$
- Rechenfunktion ϕ_d: $d' = d + b_d \mod 4$
- Rechenfunktion ϕ_τ: $\tau' = \tau$ für $b_\tau = 0$, $\tau' = leer$ für $b_\tau = 1$ und $\tau = voll$, $\tau' = voll$ für $b_\tau = 1$ und $\tau = leer$

wobei $b_\tau = 0$ bedeutet, dass der Zelltyp nicht verändert wird (keine Bewegung), und $b_\tau = 1$, dass der Zelltyp entsprechend der Bewegung des Agenten geändert wird. Die Richtungen N, E, S, W sind hier mit den Werten 0 bis 3 codiert, um die Formel für eine Drehung um 90 Grad im Uhrzeigersinn im Fall der Aktion a_0 als Modulo-Rechnung angeben zu können.

Der Eingang x entspricht den Werten, die der Agent mit seinem Sensor wahrnimmt, seine Bedeutung, und die Bedeutung der einzelnen Zustände s sind in diesem Beispiel nicht spezifiziert. Tab. 3.1 gibt die Zustandsübergangsfunktion sowie die Ausgabefunktion an. Abhängig vom momentanen Zustand und dem Input ist ein Paar vom Folgezustand s' und Ausgabe e in der Form s'/e gegeben. Die Aktionen und Basisoperationen sind nicht in der Tabelle enthalten, da sie nicht das Verhalten des Agenten bestimmen, sondern lediglich Abbildungen der Entscheidungen des Agenten sind und auf den Zustand der Zelle „angewendet“ werden, d. h. auf die Bestimmung der neuen Werte d und τ.

Für FSMs, die eine bestimmte Komplexität nicht überschreiten, sind auch grafische Darstellungen möglich und übersichtlich, z. B. als Zustandsübergangsgraph (Abb. 3.18(a)) oder als Schaltwerk mit im Speicher codierter Zustandsübergangstabelle (Abb. 3.18(b)). In letzterer

Tab. 3.1.: Tabellarische Darstellung einer Beispiel-Mealy-FSM zur Kontrolle eines Agenten. Die Einträge zeigen den Folgezustand und die Entscheidung in der Form s'/e an.

Inputs	**momentaner Zustand** s					
x	**0**	**1**	**2**	**3**	**4**	**5**
x_0	$3/e_0$	$0/e_0$	$1/e_0$	$4/e_0$	$5/e_1$	$2/e_1$
x_1	$1/e_0$	$3/e_1$	$5/e_0$	$0/e_1$	$2/e_0$	$4/e_0$

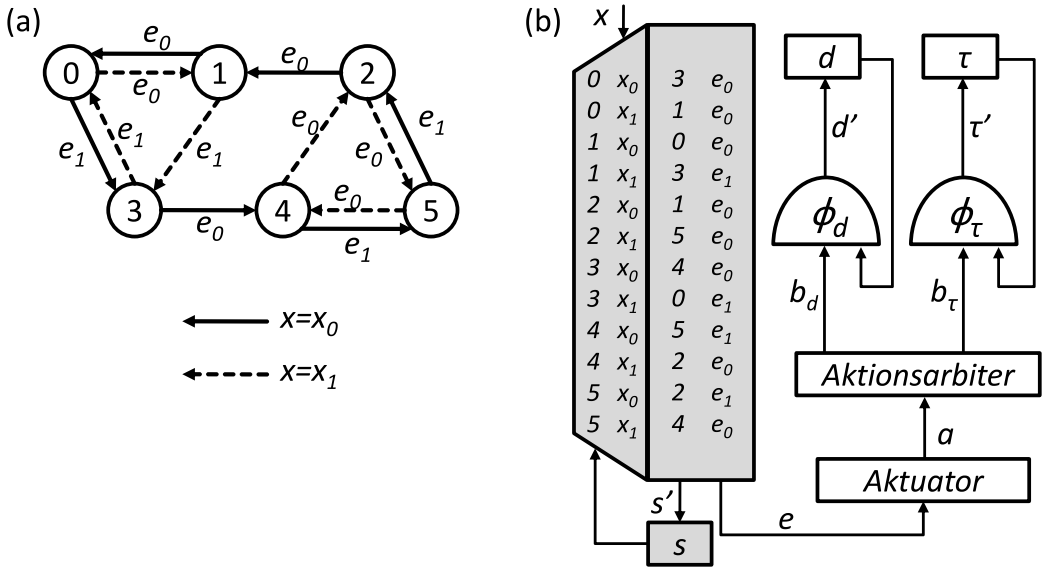

Abb. 3.18.: Graphische Darstellungen einer Beispiel-Mealy-FSM zur Kontrolle eines Agenten: (a) als Zustandsgraph, (b) als Schaltwerk mit Speicher. Die Teile der Kontroll-FSM in (b) sind schattiert.

Darstellung ist der komplette Moore-Automat bestehend aus der Mealy-FSM und den Rechenwerken gegeben. Die für das Verhalten des Agenten bestimmende Mealy-FSM ist schattiert.

Grundsätzlich ist es auch möglich, mehr als eine FSM zur Kontrolle des Agentenverhaltens einzusetzen. So könnte es einen gerichteten, beweglichen und bunten Agenten geben, der einerseits eine Mealy-FSM benutzt, um seine Bewegungen zu steuern, also letzten Endes eine Modifikation der Attribute d und τ bewirkt wie im obigen Beispiel, und andererseits eine andere Mealy-FSM benutzt, um sein Attribut *Farbe* zu steuern.

In Abschn. 3.1.2 wurde beschrieben, dass die Replikate der Kontrolleinheiten unter bestimmten Bedingungen nicht notwendig sind. Das ist dann der Fall, wenn die Regeln so beschaffen sind, dass die einzig vorhandene Kontrolleinheit die (Bewegungs-)Aktionen der Agenten aus den Nachbarzellen berechnen kann. Dann muss die Struktur um einen Multiplexer erweitert werden, der den richtigen momentanen Kontrollzustand für die Bestimmung des Folgezustands und der Entscheidung auswählt (Abb. 3.19).

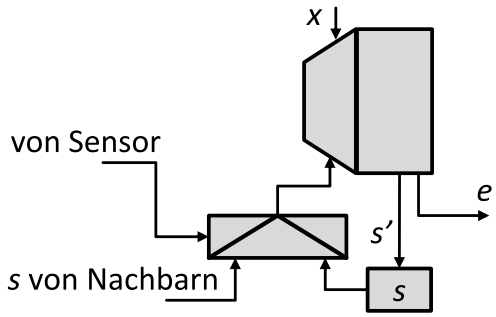

Abb. 3.19.: Zellstruktur mit Multiplexer und FSM. Darstellung als Schaltwerk mit Speichersymbol.

3.2.2 Andere Konzepte zur Modellierung des Agentenverhaltens

Im Folgenden werden einige weitere typische Konzepte (vgl. [Klü01, S. 21ff.] und [BS04]) für die Modellierung von künstlicher Intelligenz in Agentenverhalten/Agentenprogrammen beschrieben. Die hier besprochenen Konzepte können aufgrund der vielfältigen Möglichkeiten nicht vollständig und im Einzelfall nicht detailliert beschrieben sein.

Künstliche Neuronale Netze. Die Intention von *künstlichen neuronalen Netzen* (ANN, engl.: Artificial Neural Network) ist die vereinfachte Nachbildung von biologischen neuronalen Netzen, also der Versuch, die biologische Intelligenz nachzuahmen. Grundlagen von künstlichen neuronalen Netzen werden unter anderem in [RMS91, NKK96] beschrieben. Ein solches Netzwerk besteht aus vernetzten Knoten (den künstlichen Neuronen) mit gewichteten Verbindungen. Jedes Neuron besitzt eine oder mehrere eingehende Verbindungen und eine ausgehende Verbindung, welche wiederum als Eingänge für andere Neuronen fungieren können. Die Verbindungen können auch bidirektional sein. Aus den (gewichteten) Eingangswerten wird mittels einer Aktivierungsfunktion ein Ausgangswert berechnet. Dabei wird potentiell ein Schwellenwert herangezogen, ab dem der Ausgangswert sich ändert. Die Veränderung des Zustandes, d. h. aller an den Verbindungen liegenden Werte des ANN, kann durch eine synchrone oder asynchrone Neuauswertung der Funktionen der Neuronen stattfinden. Für die komplette Beschreibung eines ANN müssen die Aktivierungsfunktionen, Gewichte und Schwellenwerte, sowie die ganze Topologie des Netzes angegeben werden. Um das ANN als Verhaltenssteuerung zu benutzen, werden Eingaben von der Sensorik des Agenten und Ausgaben für den Aktuator benötigt. Dafür müssen die Neuronen schließlich in drei Schichten eingeteilt werden: Eingabeneuronen, die von der Agentensensorik Werte aufnehmen, Ausgabeneuronen, die die Schnittstelle zum Aktuator bilden, und versteckte Neuronen, die nicht mit äußeren Komponenten verbunden sind.

Üblicherweise werden ANN mit Lernverfahren eingesetzt. Das bedeutet, dass die Gewichte und Schwellenwerte adaptiert werden können, um im Falle der Verwendung des Netzes in MAS das Agentenverhalten zu optimieren. In [CJ90, CJ91] werden z. B. künstliche Ameisen mit *Genetischen Algorithmen* angelernt, um das Problem der Futtersuche effizient zu lösen. Agenten mit ANN als Verhaltenssteuerung müssen in [WM99] in unbekannten zweidimensionalen Gitternetzen mit Hindernissen navigieren. Ebenfalls in zweidimensionalen Gittern werden in [LK06] lernende Agenten eingesetzt, die sich, in zwei konkurrierende Gruppen eingeteilt, gegenseitig attackieren. Die Verhaltensfunktion der Agenten wird als ANN repräsentiert. Für ein Intrusion Detection System werden in [HCPA07] Agenten, in denen ein ANN eingebettet ist, eingesetzt. Die genannten Beispiele sind natürlich nur ein Bruchteil aller Anwendungen von ANN zur Verhaltenskontrolle, zeigen aber dessen Vielseitigkeit.

Im Gegensatz zu den FSM ist bei einem ANN der Zustand nicht explizit angegeben, sondern ergibt sich implizit aus den Werten aller Verbindungen des Netzes. Das lässt sich nicht auf triviale Weise in einen Gesamtzustand „übersetzen“, wenn die Anzahl der Verbindungen nicht klein ist. Die Interpretation des ANN als Verhaltenssteuerung ist daher nicht intuitiv und kann schwierig sein. Dennoch ist eine Anwendung im in Abschn. 3.1 entwickelten Modell denkbar. Eine Schwierigkeit, die dabei zu überwinden wäre, ist allerdings das Kopieren des Zustands auf eine Nachbarzelle bei einer Bewegung des Agenten.

Eine besondere Form der ANN sind die *rekurrenten Neuronalen Netze* [Elm90], deren Neuronen auch durch Rückkopplungen verbunden sind. Damit wird dem ANN eine Art Gedächtnis gegeben, mit dem es sich an historischen Informationen orientieren kann. Auf diese Weise kann der Kontrollfunktion ein interner Zustand gegeben werden und sogar eine zu FSMs äquivalente Funktion implementiert werden [RBA96].

Einige Wissenschaftler arbeiten auch mit *biologischen neuronalen Netzen*. Dazu werden z. B. Neuronen von Schnecken [Zec02] oder Ratten auf Siliziumchips kultiviert. In [War10] werden biologische neuronale Netze als künstliche Intelligenz trainiert, um damit z. B. fahrende Roboter zu steuern.

Fuzzylogik-Systeme. *Fuzzy Controller* können eingesetzt werden, um das Verhalten von Agenten in MAS zu steuern. In der *Fuzzylogik* werden sprachliche Konzepte, die die „scharfe“ zweiwertige Logik aufweichen und „unscharfe“ mehrwertige Logik erlauben, formalisiert. Das bedeutet, es gibt nicht nur wahr oder falsch, sondern eine graduelle Unterscheidung von Eigenschaften mit unendlich vielen Werten zwischen 0 und 1. Zugehörigkeitsfunktionen bilden Objekte auf *Fuzzymengen* ab. Dabei kann ein Objekt aufgrund der graduellen Unterscheidung seiner Eigenschaften, auf mehrere Fuzzymengen abgebildet werden, entscheidend ist dabei dessen Grad der Zugehörigkeit. Beispielsweise könnte ein Räuber-Agent mit der Eigenschaft Sättigung (wie in obigem Beispiel), gar keinen, wenig, viel oder maximalen Hunger haben. Den sprachlichen Begriffen werden Werte zwischen 0 (z. B. für gar keinen Hunger) und 1 (maximaler Hunger) zugeordnet. Ein Räuber mit viel Hunger wird dann zu einem hohen Grad auf die Menge *hungrig* und zu einem niedrigen Grad auf die Menge *satt* abgebildet. Der Grad bestimmt den Wahrheitsgehalt einer Aussage wie: *Der Räuber ist satt*. Ebenso kann mit Eigenschaften verfahren werden, die der Agent über seine Sensorik aufnimmt (z. B. hohe/niedrige Pheromonwerte).

Die Kontrolle des Agenten wird mit *Fuzzyregeln* beschrieben. Zunächst müssen die Eingangsvariablen der Kontrollfunktion *fuzzyfiziert* werden, d. h. auf jedes („scharfe“) Eingangssignal wird die Zugehörigkeitsfunktion angewendet. Ein Pheromonwert bekommt z. B. für die sprachlichen Ausdrücke wie *hoch* oder *niedrig* je einen Grad zugewiesen. Mit den sprachlichen „unscharfen“ Ausdrücken, werden dann logische Aussagen der Form *Wenn Bedingung B dann Aktion A* definiert. Das Ergebnis der Anwendung aller logischen Aussagen ist eine Menge von zutreffenden Aktionen *A*, die alle einen bestimmten *Erfüllungsgrad* (Wahrheitsgehalt) größer 0 haben. Um im MAS eine konkrete Aktion auszuführen, muss aus allen Aktionen *A* eine per *Defuzzyfizierung* ausgewählt werden. Dafür gibt es verschiedene Methoden, die meist auf Schwerpunkt-, Maxima- oder Mittelwertberechnungen basieren.

Die Hauptmotivation für die Auswahl von Fuzzy Controllern zur Agentensteuerung ist die Möglichkeit der Anwendung von sprachlichen Konzepten und Regeln, die eher den Kommunikations- und Verhaltensweisen von Menschen entspricht, als formale Strukturen. Durch die Formalisierung der sprachlichen Konzepte in Fuzzy-Systemen ist diese Möglichkeit gegeben.

Auch MAS mit Agenten, deren Verhalten von Fuzzy Controllern gesteuert wird, sind üblich und finden sich in verschiedenen Anwendungsformen. Einige Beispiele sind [BB97], wo ein evolutionärer Algorithmus vorgestellt wird, der die Fuzzyregeln von Agenten optimiert, [MZG01] und [JM08], wo ein mobiler Roboter mit einem Fuzzy Controller navigieren muss.

Classifier-Systeme. *Classifier-Systeme* [HBC+00, Lan08, Gol89] sind regelbasierte Systeme, deren Regeln parallel ausgeführt werden können und mittels Lernverfahren adaptiv verändert werden können. Daher werden sie auch als *Learning Classifier Systems* bezeichnet. Die wesentlichen Bestandteile eines typischen Classifier-Systems sind eine Regelbasis und eine Nachrichtenliste. Letztere kann eine begrenzte Menge von Nachrichten mit einer bestimmten Länge L beinhalten, die aus Wörtern (Strings) eines bestimmten Alphabets bestehen. Mit demselben Alphabet existieren Regeln der Form *Wenn Bedingung B dann Nachricht N* in der Regelbasis. Diese Regeln werden Classifier genannt, daher bezeichnet man die Regelbasis auch als Classifier-Speicher. Für die Bedingung B können außer dem eigentlichen Alphabet auch *Don't Cares* (Stellen, die bei einem Vergleich immer eine Übereinstimmung liefern) eingesetzt werden. Die Entscheidungsfindung läuft wie folgt ab. Über die Sensorik des Agenten werden Nachrichten der Länge L in die Nachrichtenliste aufgenommen. Alle Nachrichten der Liste werden dann parallel mit allen Classifiern auf Übereinstimmung mit der Bedingung B geprüft. Ist eine Bedingung erfüllt, so wird die zugehörige Nachricht N aus der entsprechenden Regel in die Nachrichtenliste übernommen. Dies kann allerdings nicht uneingeschränkt geschehen, da die Nachrichtenliste begrenzt ist. Die entstehende Konkurrenzsituation wird mit Gewichten für die Regeln und gegebenenfalls auf Zufall basierenden Auswahlverfahren für die gewichteten Regeln gelöst. Eine mehrstufige Anwendung der Regeln auf die in jeder Stufe wieder veränderte Nachrichtenliste ergibt schließlich eine Liste, aus der der Aktuator eine Aktion auswählen kann.

Typischerweise geschieht das Lernen in zweistufiger Form. Die Gewichte der Classifier werden dabei auf der niedrigeren Stufe dynamisch, also während der Agent agiert, angepasst. Dies kann mittels *verstärkendem Lernen* erfolgen, d. h. dass der Agent ein Feedback über seine ausgewählten Aktionen bekommt und die Gewichte der Regeln anpasst. Auf der höheren Stufe wird ein Genetischer Algorithmus eingesetzt, um zu bestimmten Zeitpunkten die Regeln selbst zu verändern bzw. durch neue Regeln zu ersetzen.

Ein Argument für die Verwendung von Classifier-Systemen als Kontrollfunktion ist die Möglichkeit, mehrere Regeln simultan anzuwenden. So können gleichzeitig eintreffende Eingaben als Kombination von Eingaben wahrgenommen und verarbeitet werden, anstatt sequentiell abgearbeitet zu werden. Ein weiteres Argument ist die Adaptivität des Verhaltens durch das verstärkende Lernen und die Evolvierung von Regeln.

Classifier-Systeme werden für eine Vielzahl von Anwendungen eingesetzt, darunter auch die Kontrolle von autonomen Agenten. Eine extensive Angabe von Quellen findet sich bei Lanzi in [Lan08]. In [TSTJ05] werden autonome Fahrzeuge mittels Classifier-Systemen gesteuert.

3.3 Kapitelzusammenfassung

Für die Modellierung von MAS in CA wurden *Objekte* definiert, die sich in *passive Objekte* und *Agenten* einteilen lassen. Im allgemeinen Modellierungsprinzip wurde die Einbettung des MAS in das CA-Modell mittels der Aufteilung der Transitionsfunktion und des Zustands der Zelle, sowie die Zuordnung der dadurch entstehenden Teile zu Objekten und durch Einführung eines *Zelltyps* erreicht. Die Interaktionen der Objekte im MAS innerhalb einer Zelle und interzellu-

lar wurden durch Einführung eines Aktionsarbiters ermöglicht. Ein besonderer Schwerpunkt lag in der Modellierung von *beweglichen Objekten*, da im CA-Modell bzw. in CA-Architekturen die Zellen nur lesend auf ihre Nachbarn zugreifen können. Ein konsistentes Regelpaar, welches ein bewegliches Objekt auf einer Zelle löscht und auf der anderen kopiert, umgeht dieses Problem. Die *Konfliktauflösung* kann mit einer *erweiterten Nachbarschaft* und/oder *gerichteten Agenten* innerhalb eines Taktschritts betrieben werden. Es ist ein Modell entstanden, das vom Agentenverhalten abstrahiert.

Für die Modellierung des Agentenverhaltens wurden *endliche Zustandsautomaten* (FSM) ausgewählt. Die Einbettung der FSM in die Zellstruktur unter Verwendung von *Sensoren* und *Aktuatoren* mit *Inputreduktion* und *Aktionsmapping* wurde beschrieben. Alternativen zu FSMs wurden kurz vorgestellt.

4 Beispielsysteme zur Modellierung

Mit dem in Kap. 3 vorgestellten CA-Modell werden nun drei Beispielsysteme und deren Regeln definiert. Sie sollen als problemlösende MAS dazu dienen, die Modellierungsmöglichkeiten zu veranschaulichen und außerdem Optimierungsverfahren für FSM-kontrollierte Agenten zu testen und zu entwickeln. Die drei Systeme sind:

- das *Creatures' Exploration Problem* (CEP), definiert in Abschn. 4.1
- der *All-to-All Communication Task* (ATAC), definiert in Abschn. 4.2
- das *Agent Routing Problem* (ARP), definiert in Abschn. 4.3

Jede der Anwendungen kann in verschiedenen Varianten auftreten, z. B. mit verschiedenen Attributen für die Objekte, mit verschiedenen Objekten oder mit unterschiedlich mächtigen Agenten (z. B. gerichtete und ungerichtete). Je nach Voraussetzung kann es möglich sein, das System auf diese Weise zu variieren, um die Problemlösung effektiver und effizienter zu gestalten. Die Untersuchung solcher Varianten ist Gegenstand von Kap. 5. Bestimmte Varianten aller drei Anwendungen wurden bereits in einer Reihe von Veröffentlichungen präsentiert. Beschrieben wird hier aber nur jeweils eine Grundvariante für jede Anwendung, also das prinzipielle zu lösende Problem, die Aufgabe und Art der Agenten, die Art der passiven Objekte und deren Regeln. Konkrete initiale Konfigurationen, d. h. die Feldgröße, die Anzahl der Objekte, deren Position etc. werden auch nicht spezifiziert.

Allen Anwendungen gemeinsam ist, dass sie in einem endlichen CA mit zwei Dimensionen und der von Neumann-Nachbarschaft modelliert sind. Je nach Anwendung und Variante wird jedoch ein unterschiedlicher Radius der Nachbarschaft verwendet. Der Radius ist aber so gewählt, dass bei jedem Agenten von einer lokal beschränkten Sicht auf die Umwelt gesprochen werden kann. Die Modellierung der Anwendungen ist prinzipiell auch in dreidimensionalen Gittern möglich, allerdings würde dies die Struktur und die Regeln des MAS komplexer machen.

4.1 Das Creatures' Exploration Problem

Beim *Creatures' Exploration Problem* (CEP) bewegen sich Agenten, die so genannten *Creatures*, auf einem zweidimensionalen Gitter mit Hindernissen und haben die Aufgabe, alle freien Zellen, auf denen sich keine Hindernisse befinden, mindestens einmal zu besuchen. Abgesehen davon sollen die Agenten diese Aufgabe möglichst schnell, d. h. in möglichst wenigen Generationen lösen.

Zuerst wurde das CEP mit FSM-kontrollierten Agenten in [HS03] beschrieben, allerdings mit der Einschränkung, dass sich nur ein einzelner Agent im System befindet, wodurch die Zellregeln gegenüber dem hier entwickelten allgemeinen Modell mit Konfliktlösungsmöglichkeiten einfacher sind. Zelltypen werden bereits eingeführt, um beschreiben zu können, ob sich ein Hindernis oder ein Agent in einer Zelle befindet, aber auch in dieser Hinsicht ist kein allgemeines Modell mit Objekten und deren Attributen definiert worden. Eine erste Implementierung des CEP in Hardware erfolgte in [HHHT04]. Halbach und Hoffmann führten weitere

Untersuchungen durch, mit dem Ziel, die optimale FSM zur Kontrolle einer Creature durch Aufzählung aller Möglichkeiten zu finden [HH05] und die Aufzählung selbst zu optimieren [HHB06, Hal08] und die Hardwareimplementierung in Ressourcenverbrauch und Performanz zu verbessern [HH06a, HH07a, Hal08]. Für FSMs mit bis zu sechs Zuständen konnte durch ein verbessertes Aufzählungsverfahren und parallele Simulation in optimierter Hardware ein exaktes Optimum gefunden werden. Die Effizienz von Systemen mit vielen Agenten wurde in [HH06b, HH07b] untersucht.

Anwendungen des CEP könnten autonome Rasenmäher oder Staubsauger sein, die sich in unbekanntem Gebiet mit Hindernissen (Sträucher, Bäume, Möbel, Wände) bewegen. Allgemein könnte das CEP für mobile Roboter eingesetzt werden, die eine unbekannte Umgebung sensorisch auskundschaften oder selbst eine Information (z. B. eine Chemikalie oder visuelle Markierungen) auf dem Gebiet verteilen. Eine etwas abgewandelte Form des CEP ist das Area Surveillance und Patrol Problem aus [SMVB05], in dem die Agenten ein Gebiet ebenfalls vollständig abschreiten sollen, dies allerdings mit möglichst wenigen mehrfach besuchten Stellen und eventuell kontinuierlich.

In diesem System können zwei verschiedene Objekttypen existieren: die Creatures (Agenten) und die Hindernisse (passive Objekte). Eine Creature und ein Hindernis dürfen nicht gleichzeitig auf einer Zelle sein. Es darf sich maximal eine Creature gleichzeitig auf einer Zelle befinden. Jede Creature ist beweglich, hat in der Grundvariante eine Bewegungs- und Blickrichtung $d \in \{N, E, S, W\}$ und kann die folgenden Aktionen ausführen:

- *Rs*: um 90° nach rechts drehen, ohne sich auf eine andere Zelle zu bewegen
- *Ls*: um 90° nach links drehen, ohne sich auf eine andere Zelle zu bewegen
- *Rm*: um 90° nach rechts drehen und gleichzeitig eine Bewegung auf die Zelle in (alter) Blickrichtung durchführen
- *Lm*: um 90° nach links drehen und gleichzeitig eine Bewegung auf die Zelle in (alter) Blickrichtung durchführen

Die Creature muss bei ihrer Handlung folgende Regeln befolgen: Wenn in der Zelle in Blickrichtung ein Hindernis oder eine andere Creature ist oder dort ein Konflikt auftritt, so muss die Creature auf ihrer momentanen Position verharren und darf nur die Aktionen *Rs* oder *Ls* ausführen. In allen anderen Fällen ist die Creature dazu gezwungen, sich auf die Zelle in Blickrichtung zu bewegen, sie kann also nur die Aktionen *Lm* oder *Rm* ausführen. Hier bietet sich an, die Entscheidung über die Bewegung auf eine andere Zelle mit einem Aktionsmapping im Aktuator zu modellieren, da diese Entscheidung durch die Regeln fest vorgegeben ist. Das bedeutet, die Creature kann die Entscheidungen $e \in \{R, L\}$ treffen, aus denen der Aktuator abhängig von der Situation (Konflikt) die Aktion abbildet. Dies wird in der hier präsentierten Grundvariante auch so gemacht, ist aber nicht zwingend notwendig. Man könnte ebenso der Kontroll-FSM die Entscheidung über die Bewegung überlassen und gegebenenfalls im Nachinein (im Aktuator oder Aktionsarbiter) korrigieren.

Aus dem Blickwinkel des CA-Modells betrachtet, wird der Zustand jeder Zelle z_i abgesehen von ihrer Zellstruktur, die auch die Transitionsfunktion beinhaltet, mit dem Tripel

$$(\tau_i, d_i, s_i)$$

definiert (mit dem Zelltyp τ_i, der Blickrichtung d_i und dem Kontrollzustand s_i). Der Index i kann als relative Identifikation einer konkreten Zelle verwendet werden, z. B. durch die Positionsangabe. Im Folgenden wird die Zellregel einer beliebigen Zelle z_C (Center-Zelle) beschrieben. Die Nachbarzellen von z_C werden wie in Abschn. 3.1.2 nach den Himmelsrichtungen bezeichnet. Die Zellstruktur mit Zelltyp, Objekten und deren Attributen und Kontrolleinheiten sowie die Nachbarschaft ist in Abb. 4.1 dargestellt.

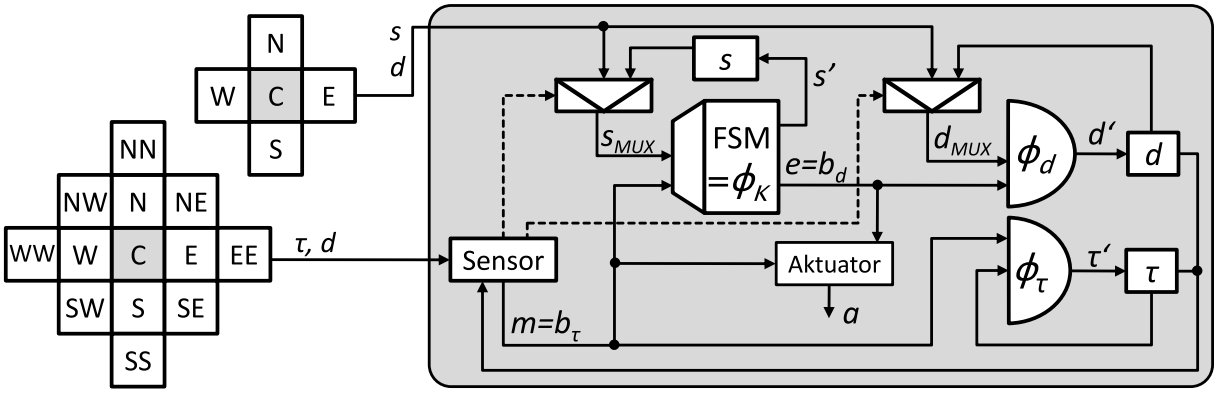

Abb. 4.1.: Zellstruktur und Nachbarschaft der Grundvariante des CEP.

Da nach den Regeln nie mehr als ein Objekt gleichzeitig auf einer Zelle sein kann, kann der Zelltyp drei verschiedene Zustände annehmen:

$$\tau \in \{Creature, Hindernis, Leer\}.$$

Das Objekt Hindernis ist ein Sonderfall eines Objekts, da es überhaupt keine Attribute besitzt. Es ist nicht beweglich und besitzt auch keine anderen Regeln. Es ist demzufolge ausschließlich im Zelltyp codiert und erfordert keine Kontrolleinheit. Hindernisse können weder entstehen noch verschwinden, d. h. sie sind auch nicht dynamisch. In einer initialen Konfiguration werden sie einmal definiert und bleiben fix auf dieser Zelle. Das bedeutet, es gibt im CEP Zellen mit dem unveränderlichem Zelltyp $\tau = Hindernis$.

Das andere Objekt, die Creature, hat eine Kontroll-FSM mit einem Zustand $s \in \{0, 1, 2, \ldots, n-1\}$. Im Folgenden wird auch von n-Zustands-FSMs gesprochen. Die *Kontrolleinheit* wird mit einer FSM modelliert, die als Input ein binäres Signal m erhält, welches vom Sensor nach einer Vorberechnung geliefert wird. m zeigt an, ob es in der Zelle in des Agenten Blickrichtung $d \in \{N, E, S, W\}$ einen Konflikt oder eine Blockade durch ein Hindernis gibt ($m = 0$) oder nicht ($m = 1$). Diese Angabe kann sich dabei je nach Zelltyp auf den Agenten, der sich gerade in der Zelle befindet oder auf einen Agenten, dessen Blickrichtung gerade auf diese Zelle zeigt, beziehen. Die Agentenrichtungen und der Zelltyp aller Zellen aus der von Neumann-Nachbarschaft mit Radius 2 werden zur Bestimmung von m im Sensor herangezogen. Die Eingangsvariablen für den Sensor sind die Variablen τ und d aus allen Nachbarzellen innerhalb des Radius 2. Für den Fall, dass die FSM in der Zelle z_C die Regel eines auf einer Nachbarzelle stationierten Agenten berechnen muss, muss die Variable s aus den Nachbarzellen innerhalb des Radius 1 als Input vorhanden sein. Über einen vom Sensor gesteuerten Multiplexer wird der von der FSM verwendete momentane Zustand s_{MUX} ausgewählt.

Die Creature besitzt nur das Attribut d. Die Aktualisierung dieses Attributs hängt nicht von der Aktion der Creature oder den Aktionen von Creatures in anderen Zellen ab, sondern nur von

deren Entscheidungen. Daher kann in dieser Zellstruktur die Entscheidung direkt als Bassioperation verwendet werden. Ein Aktionsarbiter ist nicht notwendig und der Aktuator ist lediglich dargestellt, um die Aktion im Schema „sichtbar“ zu machen. Für die Simulation des MAS ist auch der Aktuator nicht notwendig. Ein vom Sensor gesteuerter Multiplexer wählt aus, von welcher Nachbarzelle das Attribut d gegebenenfalls kopiert werden soll.

Um die Funktion des Sensors mathematisch darzustellen, werden zunächst einige Hilfsfunktionen und -variablen definiert: Mit der Hilfsfunktion h kann überprüft werden, ob in einer Zelle z_i eine Creature ist, und ob sie in eine bestimmte Richtung d blickt.

$$h(z_i, d) = \begin{cases} 1 & \text{, falls } \tau_i = \mathit{Creature} \wedge d_i = d \\ 0 & \text{, sonst.} \end{cases}$$

Die Variable h_{mL} gibt unter Benutzung dieser Information an, ob in der Zelle z_C ein Konflikt auftritt, indem sie die Anzahl der auf die Zelle z_C blickenden Agenten in den vier Nachbarzellen angibt.

$$h_{mL} = h(z_N, S) + h(z_E, W) + h(z_S, N) + h(z_W, E).$$

h_{mC} stellt einen Konflikt in der Zelle in Blickrichtung fest. Die Variable nimmt einen Wert größer 0 an, wenn in der Zelle z_C ein Agent ist und in seiner Frontzelle ein Konflikt auftritt, der durch andere Agenten mit der gleichen Frontzelle verursacht wird. Hindernisse und Agenten, die sich in der Frontzelle befinden könnten, werden nicht berücksichtigt.

$$\begin{aligned} h_{mC} =& (h(z_{NN}, S) + h(z_{NE}, W) + h(z_{NW}, E)) \cdot h(z_C, N) \\ &+ (h(z_{EE}, W) + h(z_{NE}, S) + h(z_{SE}, N)) \cdot h(z_C, E) \\ &+ (h(z_{SS}, N) + h(z_{SE}, W) + h(z_{SW}, E)) \cdot h(z_C, S) \\ &+ (h(z_{WW}, E) + h(z_{NW}, S) + h(z_{SW}, N)) \cdot h(z_C, W). \end{aligned}$$

τ_F ist der Zelltyp der Frontzelle aus Sicht des Agenten auf dieser Zelle. Der Zelltyp wird gebraucht, um festzustellen, ob in der Frontzelle ein Hindernis ist:

$$\tau_F = \begin{cases} \tau_N & \text{, falls } h(z_C, N) = 1 \\ \tau_E & \text{, falls } h(z_C, E) = 1 \\ \tau_S & \text{, falls } h(z_C, S) = 1 \\ \tau_W & \text{, falls } h(z_C, W) = 1 \\ \mathit{Leer} & \text{, sonst.} \end{cases}$$

Diese Informationen werden in Abhängigkeit des eigenen Zelltyps zur Berechnung von m benutzt:

$$m = \begin{cases} 1 & \text{, falls } (h_{mL} = 1 \wedge \tau_C = \mathit{Leer}) \vee (h_{mC} = 0 \wedge \tau_C = \mathit{Creature} \wedge \tau_F = \mathit{Leer}) \\ 0 & \text{, sonst.} \end{cases}$$

Wenn die Zelle selbst leer ist und exakt ein Agent aus einer Nachbarzelle auf sie blickt oder die Zelle einen Agenten beherbergt und in der Frontzelle kein Konflikt auftritt, dann ist eine

Bewegung möglich und die FSM bekommt als Eingabe $m = 1$. Die mathematische Formulierung für die Auswahl von s_{MUX} lautet:

$$s_{MUX} = \begin{cases} s_N & \text{, falls } h_{mL} = h(z_N, S) = 1 \wedge \tau_C = Leer \\ s_E & \text{, falls } h_{mL} = h(z_E, W) = 1 \wedge \tau_C = Leer \\ s_S & \text{, falls } h_{mL} = h(z_S, N) = 1 \wedge \tau_C = Leer \\ s_W & \text{, falls } h_{mL} = h(z_W, E) = 1 \wedge \tau_C = Leer \\ s_C & \text{, sonst.} \end{cases}$$

Die FSM gibt abhängig von den beiden Eingaben (aktueller Zustand s_{MUX} und Input m) den neuen Zustand s' und die Entscheidung $e \in \{R, L\}$ aus, während die möglichen Aktionen $a \in \{Rs, Ls, Rm, Lm\}$ sind. Die Transformation von Entscheidung in Aktion nimmt der Aktuator per Aktionsmapping wie folgt vor:

$$a = \begin{cases} Ls & \text{, falls } e = L \wedge m = 0 \\ Rs & \text{, falls } e = R \wedge m = 0 \\ Lm & \text{, falls } e = L \wedge m = 1 \\ Rm & \text{, falls } e = R \wedge m = 1. \end{cases}$$

Hierbei ist zu beachten, dass das Aktionsmapping in der Grundvariante nur die Funktion hat, die Aktion für den Entwickler des MAS auszugeben. Innerhalb der Zelle wird in dieser Modellierung die Aktion nicht weiter verwendet. Das liegt daran, dass die Bewegung von einer Zelle zu einer anderen implizit durch die Regeln vorgegeben ist, und es keine Attribute gibt, die von der Aktion abhängig sind.

Für die komplette Berechnung des neuen Zustands fehlen noch die neue Richtung und der neue Zelltyp. Zur Berechnung der neuen Richtung d' wird die Entscheidung und die alte Richtung benötigt, wobei die alte Richtung entweder von einem Agenten auf der Zelle z_C oder einer der Nachbarzellen kommt. Über einen Multiplexer wird diese Richtung d_{MUX} analog zu s_{MUX} bestimmt:

$$d_{MUX} = \begin{cases} d_N & \text{, falls } h_{mL} = h(z_N, S) = 1 \wedge \tau_C = Leer \\ d_E & \text{, falls } h_{mL} = h(z_E, W) = 1 \wedge \tau_C = Leer \\ d_S & \text{, falls } h_{mL} = h(z_S, N) = 1 \wedge \tau_C = Leer \\ d_W & \text{, falls } h_{mL} = h(z_W, E) = 1 \wedge \tau_C = Leer \\ d_C & \text{, sonst.} \end{cases}$$

Die neue Richtung des Agenten berechnet sich dann durch die Rechenfunktion ϕ_d wie folgt:

$$d' = \begin{cases} N & \text{, falls } (d_{MUX} = E \wedge e = L) \vee (d_{MUX} = W \wedge e = R) \\ E & \text{, falls } (d_{MUX} = S \wedge e = L) \vee (d_{MUX} = N \wedge e = R) \\ S & \text{, falls } (d_{MUX} = W \wedge e = L) \vee (d_{MUX} = E \wedge e = R) \\ W & \text{, falls } (d_{MUX} = N \wedge e = L) \vee (d_{MUX} = S \wedge e = R). \end{cases}$$

Schließlich wird eine Aktualisierung des Zelltyps durch die Rechenfunktion ϕ_τ vorgenommen:

$$\tau' = \begin{cases} \mathit{Hindernis} & \text{, falls } \tau_C = \mathit{Hindernis} \\ \mathit{Leer} & \text{, falls } (\tau_C = \mathit{Creature} \wedge m = 1) \vee (\tau_C = \mathit{Leer} \wedge m = 0) \\ \mathit{Creature} & \text{, falls } (\tau_C = \mathit{Leer} \wedge m = 1) \vee (\tau_C = \mathit{Creature} \wedge m = 0). \end{cases}$$

Hindernisse bleiben immer Hindernisse. Leere Zellen kopieren den Agenten und seine Attribute bzw. berechnen seine neuen Attribute (hier die Richtung), wenn eine Bewegung möglich ist. Zellen, die eine Creature beherbergen, verhalten sich umgekehrt. Wenn eine Bewegung möglich ist, wird die Creature gelöscht, indem der Zelltyp auf *Leer* gesetzt wird. Mit Abschluss aller Berechnungen werden die neuen Werte in allen Zellen gleichzeitig (also getaktet) als Zustandstripel übernommen:

$$(\tau_C, d_C, s_C) \Leftarrow (\tau', d', s').$$

Das Berechnen der Richtung d und des Zustands s für Zellen, die keinen Agenten aufgenommen haben, ist unproblematisch, da durch die Berücksichtigung des Zelltyps bei den Regeln „falsche“ Richtungen und FSM-Zustände ignoriert werden. Die Bewegung ist eine konsistente Ausführung der Löschung in der Quellzelle und des Kopierens in der Zielzelle. Diese Konsistenz wird sichergestellt durch die Berechnung von m für die eigene sowie für die Nachbarzellen.

Die Zellregel als eine ganze Einheit lässt sich auch als Pseudocode formulieren (List. 4.1). Im Pseudocode werden ausschließlich die Änderungen des Zustands beschrieben, daher ist diese Beschreibung kürzer als die mathematische. Sie liegt außerdem etwas näher an der intuitiven Vorstellung der Verarbeitung von Eingaben, da sie einfache Wenn-dann-Konstrukte enthält, die sequentiell abgearbeitet werden. Um die Bestimmung der Agentenrichtung mit einer Modulo-Rechnung durchführen zu können, werden die Entscheidungen e und die Richtungen d mit natürlichen Zahlen codiert, was eine weitere Verkürzung der Beschriebung der Zellregel ermöglicht. Dabei entspricht $L = -1$, $R = 1$, $N = 0$, $E = 1$, $S = 2$ und $W = 3$. Die beiden Methoden *FSMnewstate(s,m)* und *FSMoutput(s,m)* geben den Folgezustand bzw. die Ausgabe der FSM bei den im Argument spezifizierten Parametern zurück. Die Zuweisung $\Leftarrow$ steht für eine synchronisierte Übernahme der Werte am Ende der Regelberechnung. Auf der obersten Ebene wird der Zelltyp abgefragt (Z. 1 und 13). Ist die Zelle leer, werden sequentiell alle Nachbarzellen auf einen Agenten mit Bewegungswunsch überprüft und gegebenenfalls dessen Daten kopiert (Z. 3 bis 6). Danach wird bei zu erfolgender Bewegung die Aktion per FSM bestimmt und alle Zustandsvariablen aktualisiert (Z. 8 bis 11). Wenn in der Zelle eine Creature ist, dann wird über eine Fallunterscheidung überprüft, ob in der Frontzelle ein Konflikt auftritt (Z. 16 bis 24). Danach wird, wenn keine Bewegung stattfinden darf, der neue Zustand berechnet und gesetzt (Z. 26 bis 28), oder im Bewegungsfall der Zelltyp aktualisiert (Z. 30).

List. 4.1: Die Zellregel des CEP als Pseudocode

```
1 if (τ_C==Leer){
2   m = 0;
3   if (τ_N==Creature && d_N==S){m++; s_MUX=s_N; d_MUX=d_N}   //Agent aus Nachbar-
4   if (τ_E==Creature && d_E==W){m++; s_MUX=s_E; d_MUX=d_E}   //zelle kopieren, wenn
5   if (τ_S==Creature && d_S==N){m++; s_MUX=s_S; d_MUX=d_S}   //Bewegungswunsch
6   if (τ_W==Creature && d_W==E){m++; s_MUX=s_W; d_MUX=d_W}   //besteht
7   if (m==1){        //Konflikt überprüfen (genau ein Bewegungswunsch?)
8     e = FSMoutput(s_MUX, 1);       //Aktion bestimmen
```

```
9       s_C ⇐ FSMnewstate(s_MUX, 1);     //neuen Zustand berechnen
10      d_C ⇐ (d_MUX + e) % 4;           //neue Agentenrichtung setzen
11      τ_C ⇐ Creature;                  //Zelltyp auf Creature setzen
12    }
13  } else if (τ_C==Creature){
14    m = 1;
15    //Konflikt und Hindernis in Frontzelle überprüfen:
16    if (d_C==N){
17      if ((d_NN==S && τ_NN==Creature)||(d_NE==W && τ_NE==Creature)||(d_NW==E && τ_NW
          ==Creature)||τ_N!=Leer) m = 0;
18    } else if (d_C==E){
19      if ((d_EE==W && τ_EE==Creature)||(d_NE==S && τ_NE==Creature)||(d_SE==N && τ_SE
          ==Creature)||τ_E!=Leer) m = 0;
20    } else if (d_C==S){
21      if ((d_SS==N && τ_SS==Creature)||(d_SE==W && τ_SE==Creature)||(d_SW==E && τ_SW
          ==Creature)||τ_S!=Leer) m = 0;
22    } else if (d_C==W){
23      if ((d_WW==E && τ_WW==Creature)||(d_NW==S && τ_NW==Creature)||(d_SW==N && τ_SW
          ==Creature)||τ_W!=Leer) m = 0;
24    }
25    if (m==0) {
26      e = FSMoutput(s_C, 0);           //Aktion bestimmen
27      s_C ⇐ FSMnewstate(s_C, 0);       //neuen Zustand berechnen
28      d_C ⇐ (d_C + e) % 4;             //neue Agentenrichtung setzen
29    } else {
30      τ_C ⇐ Leer;                      //Agent löschen
31  }
```

Je nach Art der Implementierung des CEP als Hardwaresystem oder in Software, kann die hier mathematisch und als Pseudocode beschriebene Zellregel auch in Hardwarebeschreibungssprachen (eine Beschreibung der Hardware einer Zelle in Verilog HDL befindet sich in List. A.1) oder üblichen Programmiersprachen beschrieben werden. Durch die taktbasierte Aktualisierung der Zustände, von denen wiederum der neue Zustand abhängt, wird die Zellregel in Hardware durch ein Schaltwerk realisiert. Ohne die Aktualisierung erhält man ein kombinatorisches Schaltnetz ohne Rückkopplungen. Der komplette Schaltplan mit einfachen Gattern, Vergleichern und Multiplexern ist in Abb. 4.2 dargestellt.

Der Aktuator ist, da er keine Bedeutung bei der Berechnung der Folgezustände hat, kein Teil der tatsächlich benötigten Hardware. Den anderen Komponenten (zwei Rechenwerke, die FSM, die Multiplexer (MUX) und der Sensor) sind die einzelnen Gatter durch farbliche Unterlegung zugeordnet. Die Register zum Speichern des Zelltyps, der Richtung und des Kontrollzustands sind mit dem Taktsignal *CLK* getaktet.

Die Werte der Richtungen entsprechen der Codierung im Pseudocode. Auf diese Weise kann im Rechenwerk für d mit einer simplen Addition die neue Richtung d' bestimmt werden. Bei dem Register handelt es sich um ein 2 Bit breites Register. Bei Ergebnissen der Addition, die mehr Bits benötigen, werden diese einfach ignoriert, so dass eine Addition mit 3, einer Subtraktion mit 1 entspricht. Eine Fallunterscheidung, wie bei der Angabe als mathematische Formel oder eine Modulo-Rechnung wie im Pseudocode ist daher nicht notwendig. Die Werte für den Zelltyp sind abkgekürzt. Dabei steht *Le* für *Leer*, *Cr* für *Creature* und *Hi* für *Hindernis*. Um eine bessere Übersicht zu erreichen, sind einige mehrfach verwendete Konstanten und Inputs von den Nachbarzellen nicht verbunden dargestellt. Das bedeutet nicht, dass diese Konstanten oder Inputs mehrfach vorhanden sein müssen. Die Multiplexer zur Auswahl der Richtung und des Kontroll-

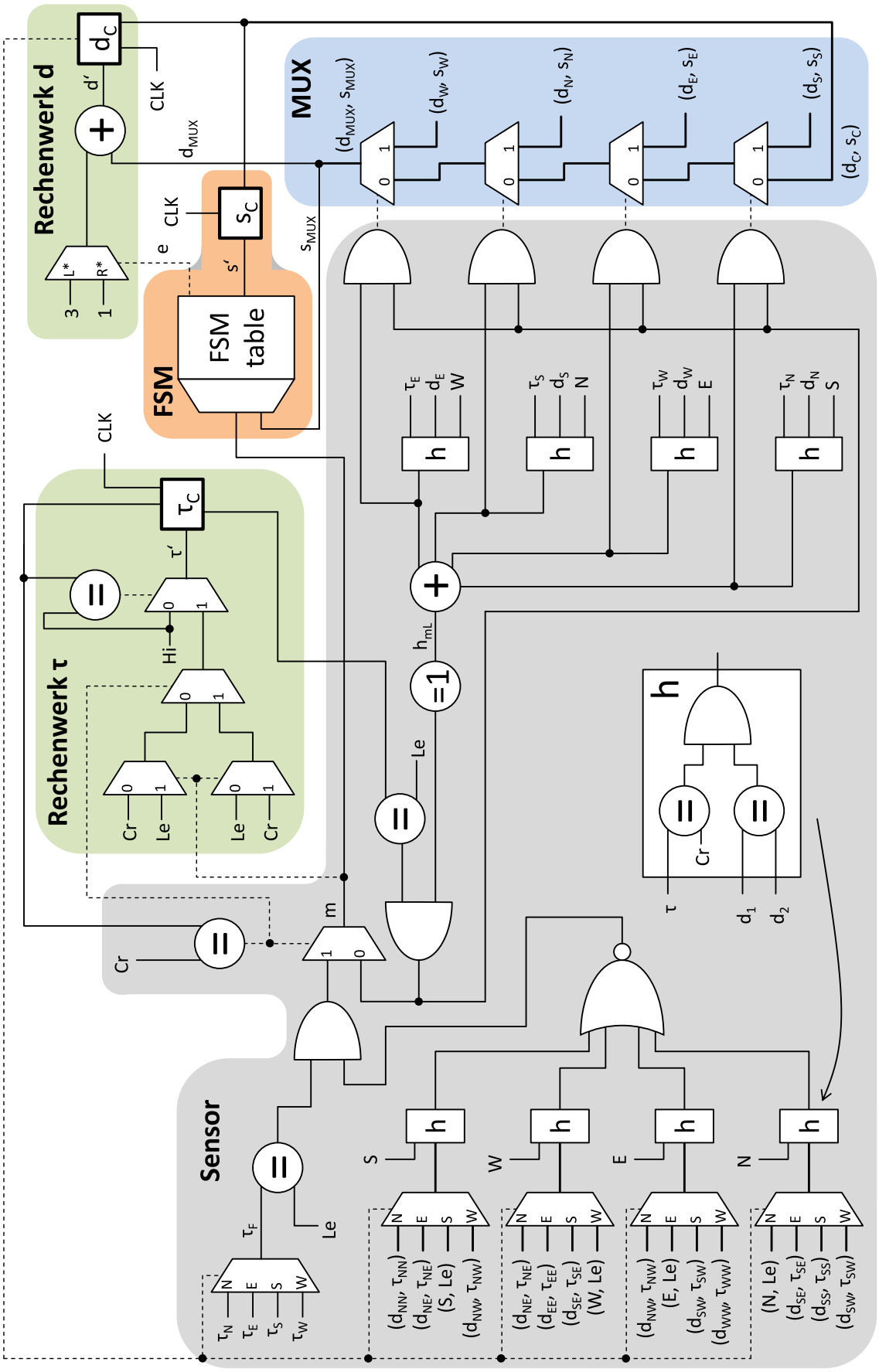

Abb. 4.2.: Hardwareschaltplan einer Zelle im CEP.

zustandes sind hier als vier hintereinander geschaltete Multiplexer realisert. Es wäre ebenfalls möglich, die vier einzelnen Kontrollsignale auf ein einziges abzubilden, und einen Multiplexer mit fünf Eingängen zu verwenden. Weitere Variationen zur Implementierung in Hardware sind denkbar. Beispielsweise ist es möglich, je zwei der h-Module im Sensor zusammenzufassen und die ansteuernden Multiplexer zu erweitern (Eingänge und Kontrollsignal).

Abb. 4.3 zeigt eine Sequenz von Simulationsbildern einer bestimmten Konfiguration des CEP. In dieser Konfiguration ist der Rand des Feldes mit Hindernissen modelliert. Dies erspart eine andere spezielle Randbehandlung, wenn man nicht mit Wrap-Around arbeiten will. Die vier gerichteten Agenten lösen das Exploration Problem darin in der 34. Generation. Alle Agenten sind homogen (gleiche Kontroll-FSM in jeder Zelle). Die Zustandsübergangstabelle der FSM, die dieses Verhalten auslöst, ist in Tab. 4.1 gegeben. In der Abbildung sind die bereits besuchten Zellen schattiert dargestellt. Je dunkler, desto öfter wurde die Zelle besucht. Diese Angabe ist nicht Teil des MAS, sondern dient nur dem Entwickler für Analysezwecke, d. h. die Agenten können diese Information nicht wahrnehmen und sie ist deshalb auch nicht in der Zellstruktur und Zellregel enthalten. Die Feststellung der Lösung des Problems muss also global geschehen. Die Agenten haben keine eigene Abbruchbedingung.

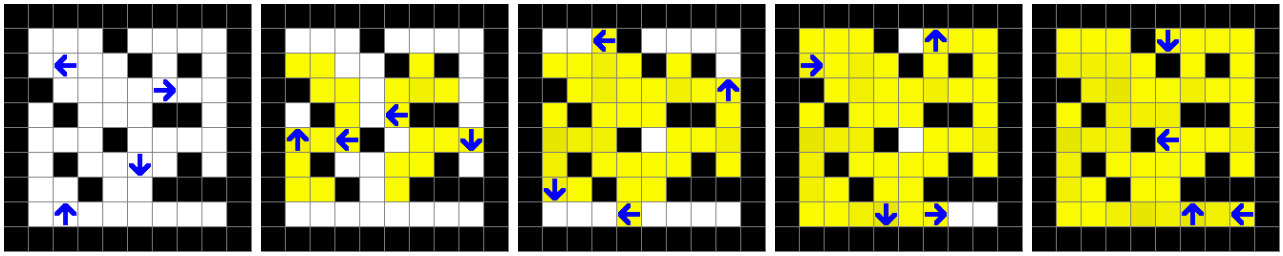

Abb. 4.3.: Sequenz einer Simulation des CEP. Hindernisse sind als schwarze Zellen dargestellt. Die bereits besuchten Zellen sind schattiert, je dunkler desto öfter wurde die Zelle besucht. Der jeweilige Zustand s der Kontroll-FSM ist nicht dargestellt.

Tab. 4.1.: Zustandsübergangstabelle der Kontroll-FSM für die Beispielsequenz des CEP.

	$s=0$	$s=1$	$s=2$	$s=3$	$s=4$
$m=0$	$1/L$	$3/L$	$4/R$	$4/L$	$2/L$
$m=1$	$4/L$	$0/R$	$0/R$	$3/R$	$1/R$

Bei der Modellierung einer Variante des CEP müssen die Regeln und die Struktur konsistent angepasst werden. Sollen z. B. ungerichtete Agenten modelliert werden, fällt aus der Struktur die Richtung d weg. Die Aktionen der FSM können sinnvollerweise keine Drehungen mehr auslösen, und die Konfliktbehandlung wird komplexer. Sogar die Nachbarschaft ändert sich.

4.2 Der All-to-All Communication Task

Die Aufgabe der Agenten beim All-to-All Communication Task (ATAC) ist, eine initial disjunkt verteilte Information mit allen anderen Agenten gegenseitig auszutauschen. Das bedeutet, jeder

Agent besitzt zu Anfang einen Teil der Information, den sonst kein anderer Agent besitzt. Das Ziel ist, dass am Ende alle Agenten die komplette Information besitzen. Der Austausch von Informationen findet dabei implizit (also ohne die aktive Auslösung durch eine Agentenaktion) in bestimmten Situationen statt, in denen sich die gerichteten Agenten auf einem zweidimensionalen Gitter durch ihre Bewegungsaktionen räumlich anordnen. Die Situationen werden auch *Kommunikationssituationen* genannt und sind ganz bestimmte lokale Muster (Abb. 4.4). Die Muster entsprechen den Mustern der Konfliktsituationen, in denen mehrere Agenten auf die gleiche Frontzelle zeigen. Voraussetzung für den Austausch der Informationen unter allen beteiligten Agenten ist, dass die Frontzelle nicht durch einen anderen Agenten oder ein anderes Objekt belegt ist. Die Frontzelle wird auch *Mediatorzelle* genannt, da sie als ein Mediator für die Kommunikation der benachbarten Agenten interpretiert werden kann. Eine von einem Agenten einmal aufgenommene fremde Information wird bei der nächsten Kommunikationssituation weiter propagiert, so dass zur Lösung des Problems nicht zwangsläufig jeder Agent mit jedem anderen Agent kommunizieren muss. Die Aufgabe soll auch hier möglichst schnell, d. h. in wenigen Generationen gelöst werden.

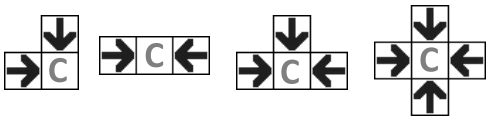

Abb. 4.4.: Muster der Kommunikationssituationen im ATAC. Zwei, drei oder vier Agenten können an der Kommunikation beteiligt sein. Die Ausrichtung des Gesamtmusters (Rotation) spielt keine Rolle. Eine Mediatorzelle ist mit *C* markiert.

Zuerst wurde das ATAC in [EH08b] beschrieben, allerdings nicht in der hier vorgestellten Grundvariante, sondern mit anderen Kommunikationsmustern. In der Grundvariante wurde das Problem zum ersten Mal in [HE08] untersucht. In Anlehnung an das CEP wurden die Agenten dort auch als Creatures bezeichnet. Die Wörter *Creature* und *Agent* für den Zelltyp sind äquivalent. Ein ähnliches System wie das ATAC ist das *Rendezvous Problem*, bei dem sich alle Agenten eines MAS gemeinsam an einem Punkt treffen müssen [LMA05].

Anwendungen des ATAC sind Systeme mit Agenten, die sich synchronisieren müssen, eine verteilte Information zusammenfügen und allen bereit stellen müssen oder einen Konsens finden müssen. Andere Systeme, in denen verteilte Agenten einen Konsens finden müssen, werden in [OFM07] besprochen.

Die Verbreitung der Information wird über einen *Kommunikationsvektor* $\vec{c}$ in jedem Agenten modelliert. Im System befinden sich k Agenten. Jeder Agent hat einen Kommunikationsvektor mit k Komponenten. Jede Komponente kann die binären Werte 0 oder 1 annehmen, weshalb auch von einem Bitvektor gesprochen werden kann. Der Kommunikationsvektor des Agenten mit dem Index j wird initial an Stelle j auf 1 gesetzt, an allen anderen Stellen auf 0. Treffen zwei oder mehr Agenten in einer Kommunikationssituation zusammen werden die Kommunikationsvektoren aller beteiligten Agenten komponentenweise (bitweise) verodert. Abb. 4.5 zeigt einen beispielhaften Ablauf der Veränderung des Kommunikationsvektors zusammen mit der zugehörigen Simulationssequenz. Die Markierungen der bereits besuchten Felder und der Zellen, die als Mediatorzelle fungiert haben sind für die Agenten nicht sichtbar, sondern nur in der Simulation dargestellt für die Analyse des Agentenverhaltens. Da die Agenten jeweils nur ihren eigenen Kommunikationsvektor und gegebenenfalls den ihrer Nachbarn lesen können, muss die Abbruchbedingung global festgestellt werden.

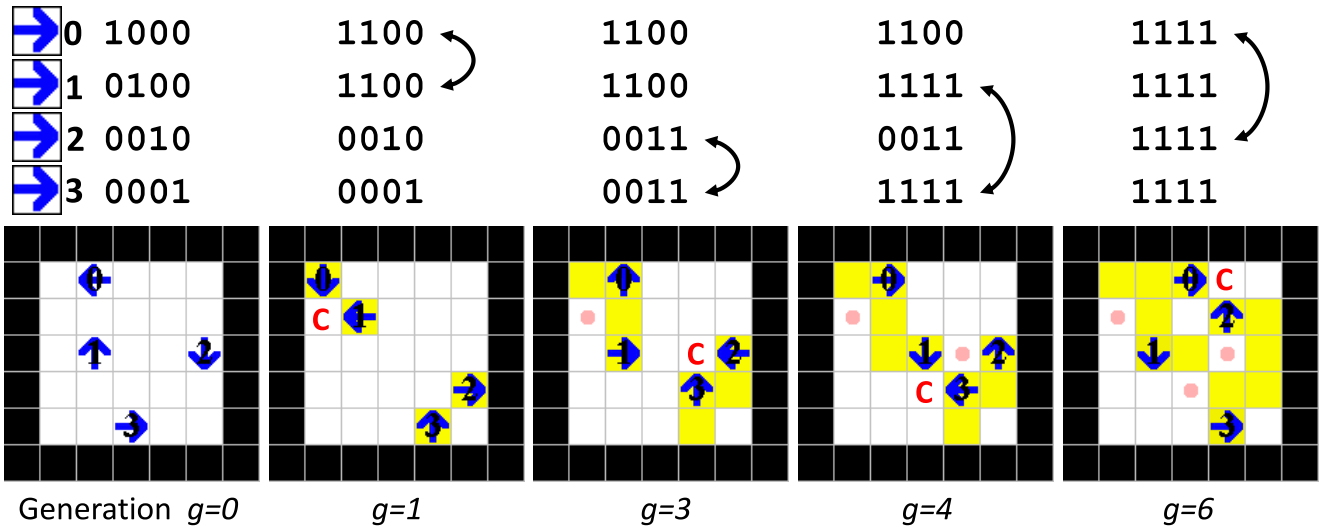

Abb. 4.5.: Simulationssequenz des ATAC mit Verbreitung von Informationen. Oben sind die Kommunikationsvektoren der Agenten 0 bis 3 angegeben. Bereits von Agenten besuchte Felder sind schattiert. Zellen, die bereits als Mediator fungiert haben, sind mit einem Kreis markiert.

Beim ATAC in der Grundvariante gibt es zwei Typen von Objekten, die Agenten und die Hindernisse. Die Bewegung der Agenten wird genauso wie beim CEP modelliert, daher werden die entsprechenden Formeln an dieser Stelle nicht nochmal angegeben, sondern nur die zusätzlichen Formeln und die erweiterte Zellstruktur (Abb. 4.6), die für die Realisierung der Kommunikation notwendig ist. Die Zellstruktur muss gegenüber dem CEP um den Kommunikationsvektor erweitert werden, wodurch sich der Zustand der Zelle über ein Quadrupel definiert:

$$(\tau_i, d_i, s_i, \vec{c}_i).$$

Der Kommunikationsvektor ist unabhängig von der gewählten Bewegungsaktion des Agenten durch die implizite Kommunikationsregel bestimmt, daher kann die Berechnung des neuen Zustands komplett im Sensor erfolgen.

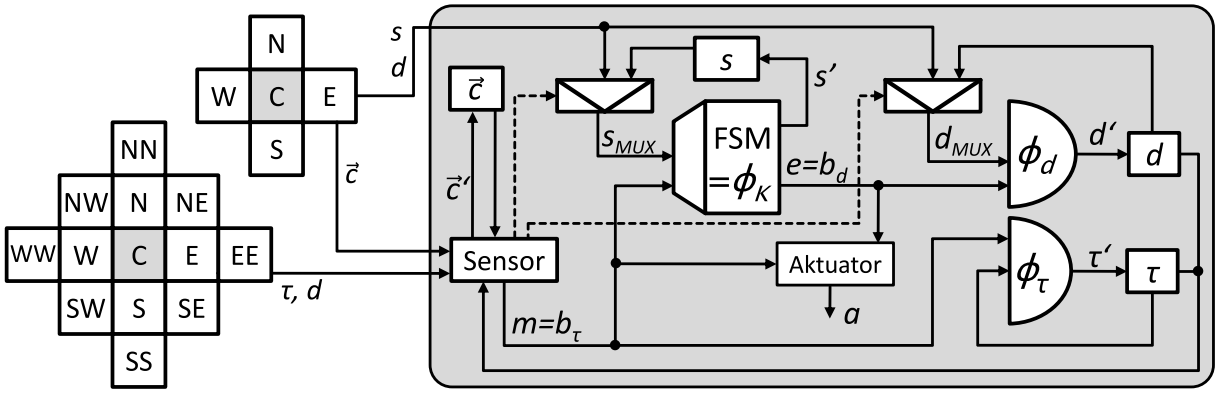

Abb. 4.6.: Zellstruktur und Nachbarschaft der Grundvariante des ATAC.

Die Aktualisierung der Variablen τ_C, d_C und s_C in der Zelle z_C entspricht der Beschreibung aus Abschn. 4.1. Der neue Zustand des Kommunikationsvektors kann entweder unverändert sein

($\vec{c}_C$), eine Kopie aus der Nachbarzelle ($\vec{c}_{COP}$) durch einen sich bewegenden Agenten oder der veroderte Vektor bei einer Kommunikation ($\vec{c}_O$). Zunächst werden einige Hilfsvariablen definiert. $\vec{c}_{COP}$ ist die Kopie des Kommunikationsvektors aus der Nachbarzelle, wenn dort ein Agent auf diese Zelle blickt:

$$\vec{c}_{COP} = \begin{cases} \vec{c}_N & \text{, falls } h_{mL} = h(z_N, S) = 1 \wedge \tau_C = Leer \\ \vec{c}_E & \text{, falls } h_{mL} = h(z_E, W) = 1 \wedge \tau_C = Leer \\ \vec{c}_S & \text{, falls } h_{mL} = h(z_S, N) = 1 \wedge \tau_C = Leer \\ \vec{c}_W & \text{, falls } h_{mL} = h(z_W, E) = 1 \wedge \tau_C = Leer \\ \vec{c}_C & \text{, sonst.} \end{cases}$$

Ein komponentenweises Maximum wird gebildet, um die einzelnen Informationen zu verteilen, sprich auf $\vec{c}_O$ zu übertragen:

$$\vec{c}_O = \begin{cases} \max(\vec{c}_C, \vec{c}_{NN} \cdot h(z_{NN}, S), \vec{c}_{NE} \cdot h(z_{NE}, W), \vec{c}_{NW} \cdot h(z_{NW}, E)) & \text{, falls } h(z_C, N) = 1 \\ \max(\vec{c}_C, \vec{c}_{EE} \cdot h(z_{EE}, W), \vec{c}_{NE} \cdot h(z_{NE}, S), \vec{c}_{SE} \cdot h(z_{SE}, N)) & \text{, falls } h(z_C, E) = 1 \\ \max(\vec{c}_C, \vec{c}_{SS} \cdot h(z_{SS}, N), \vec{c}_{SE} \cdot h(z_{SE}, W), \vec{c}_{SW} \cdot h(z_{SW}, E)) & \text{, falls } h(z_C, S) = 1 \\ \max(\vec{c}_C, \vec{c}_{WW} \cdot h(z_{WW}, E), \vec{c}_{NW} \cdot h(z_{NW}, S), \vec{c}_{SW} \cdot h(z_{SW}, N)) & \text{, falls } h(z_C, W) = 1 \\ \vec{c}_C & \text{, sonst.} \end{cases}$$

Eine Aktualisierung des Kommunikationsvektors durch Veroderung kann nur geschehen, wenn sich der Agent nicht bewegt und die Frontzelle leer ist. Falls die Zelle selbst leer ist, kann die Kopie (die ja eventuell auch der alten Information entspricht) übernommen werden. In allen anderen Fällen bleibt der Vektor unverändert:

$$\vec{c}' = \begin{cases} \vec{c}_{COP} & \text{, falls } \tau_C = Leer \\ \vec{c}_O & \text{, falls } h_{mC} \geq 1 \wedge \tau_F = Leer \\ \vec{c}_C & \text{, sonst.} \end{cases}$$

Schließlich wird der neue Zustand der Zelle übernommen:

$$(\tau_C, d_C, s_C, \vec{c}_C) \Leftarrow (\tau', d', s', \vec{c}').$$

Die Zellregel inklusive der Bewegung der Agenten kann als Pseudocode formuliert werden (List. 4.2). Die Codierung der Richtung und der bedingten Aktionen erfolgt wie in Abschn. 4.1. Die Operation | ist ein bitweises/komponentenweises Oder. Auf der obersten Ebene findet die Abfrage des Zelltyps statt (Z. 1 und 14). Bei leeren Zellen wird analog zum CEP verfahren (Bewegungswunsch feststellen, eventuell kopieren und Zustände aktualisieren, Z. 3 und 12). Ist in der Zelle ein Agent, wird per Fallunterscheidung die Blickrichtung überprüft, und dann sequentiell alle möglichen Kommunikationssituationen und Konflikte überprüft und jeweils gegebenenfalls der Kommunikationsvektor $\vec{c}_O$ mit dem des anderen Agenten verodert (Z. 17 bis 37). Danach wird im Falle der Bewegung der Zelltyp auf *leer* gesetzt (Z. 44) oder im anderen Fall die neuen Zustände berechnet und aktualisiert (Z. 39 bis 42).

List. 4.2: Die Zellregel des ATAC als Pseudocode

```
if (τ_C==Leer){
  m = 0;
  if (τ_N==Agent && d_N==S){m++; s_MUX=s_N; d_MUX=d_N; c⃗_COP=c⃗_N} //Agent aus Nach-
  if (τ_E==Agent && d_E==W){m++; s_MUX=s_E; d_MUX=d_E; c⃗_COP=c⃗_E} //barzelle kopieren
  if (τ_S==Agent && d_S==N){m++; s_MUX=s_S; d_MUX=d_S; c⃗_COP=c⃗_S} //wenn Bewegungs-
  if (τ_W==Agent && d_W==E){m++; s_MUX=s_W; d_MUX=d_W; c⃗_COP=c⃗_W} //wunsch besteht
  if (m==1){      //Konflikt überprüfen (genau ein Bewegungswunsch?)
    e = FSMoutput(s_MUX, 1);   //Aktion bestimmen
    s_C ⇐ FSMnewstate(s_MUX, 1); //neuen Zustand berechnen
    d_C ⇐ (d_MUX + e) % 4;       //neue Agentenrichtung setzen
    τ_C ⇐ Agent;                 //Zelltyp auf Agent setzen
    c⃗_C ⇐ c⃗_COP;                 //Kommunikationsvektor von Creature übernehmen
  }
} else if (τ_C==Agent){
  m = 1; τ_Z = Leer; c⃗_O = c⃗_C;
  //Konflikt überprüfen und gegebenenfalls Kommunikationsvektor verodern:
  if (d_C==N){
    τ_Z = τ_N; if (τ_Z!=Leer) m=0;
    if (d_NN==S && τ_NN==Agent){m = 0; c⃗_O = c⃗_O | c⃗_NN}
    if (d_NE==W && τ_NE==Agent){m = 0; c⃗_O = c⃗_O | c⃗_NE}
    if (d_NW==E && τ_NW==Agent){m = 0; c⃗_O = c⃗_O | c⃗_NW}
  } else if (d_C==E){
    τ_Z = τ_E; if (τ_Z!=Leer) m=0;
    if (d_EE==W && τ_EE==Agent){m = 0; c⃗_O = c⃗_O | c⃗_EE}
    if (d_NE==S && τ_NE==Agent){m = 0; c⃗_O = c⃗_O | c⃗_NE}
    if (d_SE==N && τ_SE==Agent){m = 0; c⃗_O = c⃗_O | c⃗_SE}
  } else if (d_C==S){
    τ_Z = τ_S; if (τ_Z!=Leer) m=0;
    if (d_SS==N && τ_SS==Agent){m = 0; c⃗_O = c⃗_O | c⃗_SS}
    if (d_SE==W && τ_SE==Agent){m = 0; c⃗_O = c⃗_O | c⃗_SE}
    if (d_SW==E && τ_SW==Agent){m = 0; c⃗_O = c⃗_O | c⃗_SW}
  } else if (d_C==W){
    τ_Z = τ_W; if (τ_Z!=Leer) m=0;
    if (d_WW==E && τ_WW==Agent){m = 0; c⃗_O = c⃗_O | c⃗_WW}
    if (d_NW==S && τ_NW==Agent){m = 0; c⃗_O = c⃗_O | c⃗_NW}
    if (d_SW==N && τ_SW==Agent){m = 0; c⃗_O = c⃗_O | c⃗_SW}
  }
  if (m==0) {
    if (τ_Z==Leer) c⃗_C ⇐ c⃗_O;
    e = FSMoutput(s_C, 0);       //Aktion bestimmen
    s_C ⇐ FSMnewstate(s_C, 0);   //neuen Zustand berechnen
    d_C ⇐ (d_C + e) % 4;         //neue Agentenrichtung setzen
  } else {
    τ_C ⇐ Leer;                  //Agent löschen
}
```

4.3 Das Agent Routing Problem

Beim *Agent Routing Problem* (ARP) ist jedem Agenten eine *Zielposition* in der Welt zugeordnet, die er von seiner *Startposition* aus möglichst schnell erreichen soll. Wenn alle Agenten ihre Zielposition erreicht haben, ist das globale Ziel des ARP erreicht. Die Zielposition ist nicht zu

verwechseln mit der Zielzelle, die der direkten Nachbarzelle entspricht, auf die sich ein Agent in einem Taktschritt von seiner aktuellen Position aus bewegt.

Das ARP wurde zuerst in [EH09b] präsentiert. Vier Varianten wurden vorgestellt, von denen zwei auf zufälligem Verhalten der Agenten basieren und zwei auf mit FSM-kontrollierten Agenten. Jeweils eine Variante wurde mit gerichteten und eine mit ungerichteten Agenten entwickelt. Die hier vorgestellte Grundvariante des ARP ist die mit gerichteten, FSM-kontrollierten Agenten. In der Grundvariante wird jeder Agent, der auf seiner Zielposition ankommt, umgehend gelöscht. Dieses System beinhaltet daher dynamische Agenten. Weitere Bedingungen sind, dass das Ziel der Agenten nicht mit deren Startposition übereinstimmt. Noch strenger kann gefordert werden, dass die Startpositionen eines Agenten mit keiner der Zielpositionen irgendeines Agenten übereinstimmt oder umgekehrt, dass alle Startpositionen auch gleichzeitig als Zielposition fungieren. Auf die Zellregel oder die Zellstruktur hat dies jedoch keinen Einfluss. Die Abbruchbedingung wird implizit durch das Löschen der Agenten festgestellt, d. h. nach der Lösung des Problems ist einfach kein Agent mehr vorhanden. In anderen Varianten, in denen die Agenten nicht gelöscht werden, muss die Abbruchbedingung global festgestellt werden.

Die Agenten kann man sich auch als mobile Nachricht vorstellen, die von einem Ausgangspunkt möglichst schnell zu ihrem Ziel gelangen soll. Da Agenten nur gelöscht, jedoch nicht erstellt werden können, ist nach [BHW01] (*„All packets are injected at time zero“*) das ARP ein *statisches* Routing. Ein *dynamisches* Routing (*„Nodes may inject packets into the network repeatedly over a long duration“* [BHW01]) könnte implementiert werden, indem man an den Startpositionen der Agenten spezielle Objekte definiert, die einen Agenten inklusive dessen Zielposition zu bestimmten Zeitpunkten generieren.

Das Routing, das die Agenten durchführen, ist *adaptiv*, da sie abhängig von den Konflikten und anderen Inputs nicht immer den gleichen Weg wählen. Konflikte, die durch andere Agenten entstehen, können durch eine geeignete Kontroll-FSM aufgelöst werden. Durch die Adaptivität können in bestimmten Situationen auch Deadlocks verhindert werden, die bei *deterministischem* Routing durch sich gegenseitig blockierende Agenten entstehen könnten (z. B. beim XY-Routing [MOMC04]). Andere adaptive Routingtechniken mit beweglichen Agenten/Nachrichten sind unter anderem in [BHW01, DD97, DV07] präsentiert worden.

Ein CA-basierter Pfadplanungsalgorithmus für MAS wurde in [TJA08] präsentiert. Darin haben alle Agenten dieselbe Zielposition und es wird ein Schema von Richtungen oder Werten für das gesamte Feld berechnet, das den Agenten den schnellsten möglichst kollisionsfreien Weg anzeigt. Ein Algorithmus für die Suche von beweglichen Zielen wurde in [KLS07, LP09] vorrangig für Computerspieleagenten entwickelt. Die Ziele in dem hier definierten MAS sind nicht beweglich. Andere Anwendungsgebiete neben Computerspielen sind Simulationen für Evakuierungsprozesse und Transportwege oder der Transport von Nachrichten in einem Netzwerk. Das ARP kann zur Kommunikation zwischen Prozessoren auf einem Chip oder anderen Netzwerken eingesetzt werden.

Die Agentenwelt des ARP ist ein zweidimensionales Gitter mit kartesischen Koordinaten. Die Objekte im MAS sind Agenten und Hindernisse. Der Agent kennt keine absoluten Koordinatenwerte seiner Position oder der Position seines Ziels, stattdessen kennt er initial den horizontalen wie vertikalen Abstand zu seiner Zielposition $(\Delta x, \Delta y)$, welche er bei seinen Entscheidungen berücksichtigt. Bei seinen Bewegungsaktionen muss er die Abstandswerte aktualisieren. Die Bewegung selbst ist ähnlich der Bewegung des CEP und ATAC modelliert, allerdings sind die Möglichkeiten des Agenten erweitert. Er besitzt wie oben eine Blick- und Bewegungsrichtung $d \in \{N, E, S, W\}$, aber seine Aktionen sind

- *Rs*, *Ls*, *Rm*, *Lm* wie im CEP und ATAC
- *Ss*: keine Drehung (*S* wie *Straight*), keine Bewegung auf Zielzelle
- *Bs*: um 180° drehen (*B* wie *Back*), ohne sich auf eine andere Zelle zu bewegen
- *Sm*: keine Drehung, aber eine Bewegung auf die Zelle in Blickrichtung durchführen
- *Bm*: um 180° drehen und gleichzeitig eine Bewegung auf die Zelle in (alter) Blickrichtung durchführen

Wie in den beiden anderen beschriebenen Beispielsystemen, ist die Bewegung des Agenten auf eine andere Zelle implizit in den Regeln enthalten, und unterliegt nicht der Kontrollfunktion des Agenten. Die Entscheidungen des Agenten sind $e \in \{R, L, S, B\}$. Die Konfliktbehandlung, also die Feststellung, ob eine Bewegung möglich ist (und damit stattfindet) ist allerdings gegenüber den anderen beiden Systemen erweitert. Auf einer Zelle dürfen sich maximal ein Agent oder ein Hindernis befinden. Zellen in Blickrichtung, in denen sich ein Hindernis oder ein anderer Agent befindet, sind für den Agenten blockiert, es sei denn der Agent in der Frontzelle blickt zurück in Richtung Quellzelle. In letzterem Fall wird ein Swap (s. Abschn. 3.1.3) erzwungen. Freie Zellen, in denen ein Konflikt auftritt, sind für einen der beteiligten Agenten zugänglich. In jeder Zelle befindet sich eine Prioritätenliste P, die festlegt, in welcher Reihenfolge die Agenten aus der nördlichen, östlichen, südlichen und westlichen Richtung Vorrang haben. Die auf die Konfliktzelle blickenden Agenten beziehen die Prioritätenliste dann in ihre Berechnung des Wertes m mit ein. Es gibt 24 mögliche Permutationen der Prioritätenliste, von denen eine initial in der Zelle gespeichert ist und nicht mehr verändert werden kann. Insgesamt ergibt sich damit der Zustand einer Zelle als 6-Tupel

$$(\tau_i, d_i, s_i, \Delta y_i, \Delta x_i, P_i).$$

Die FSM in dieser Variante verwendet einen Input, der dem Agenten eine Information darüber gibt, in welchem Segment der Welt (relativ vom Agenten aus gesehen) seine Zielposition liegt. Dieser Input wird aus den Werten Δx, Δy und d gebildet. Die Anzahl der der möglichen Kombinationen dieser drei Inputs ist jedoch sehr groß und hängt von der Größe des Feldes ab. Eine FSM mit vielen Inputs wird sehr komplex. Daher werden diese Inputs nach einer bestimmten Funktion reduziert. In diesem Fall kann die FSM sechs verschiedene reduzierte Inputwerte x_r verarbeiten. Tab. 4.2 gibt an, welche Inputs wie reduziert werden. Die Semantik dieser Reduktion im Sinne von Zuordnung der Zielposition in Segmente ist in Abb. 4.7 dargestellt. Falls die Zielposition aus Sicht des Agenten rechts vorne oder links vorne liegt, liefert die Inputreduktion den Wert 5, ist sie vorne geradeaus erreichbar, wird eine 4 geliefert, rechts oder links vom Agenten bedeutet eine 3 als Input für die FSM, hinten links oder rechts versetzt eine 2, genau hinter dem Agenten eine 1 und schließlich eine 0, wenn der Agent bereits auf seiner Zielposition ist. Diese Situation ist allerdings in der Grundvariante ohne Bedeutung, da der Agent sich in dem Fall bereits gelöscht hat. In der Grundvariante werden die Distanzwerte, die aktuell im Agenten gespeichert sind, verwendet. Denkbar ist auch eine andere Variante, die die Distanzen so verändert, als wären sie von der Frontzelle aus berechnet worden. Dann ergibt auch die Unterscheidung des Segments 0 einen Sinn. So wie die Inputreduktionstabelle codiert ist, kann der Agent nicht zwischen rechts und links unterscheiden. Trotzdem soll seine Kontrolleinheit mit der FSM entscheiden, in welche Richtung er sich dreht. Das und die Tatsache, dass in einem System mit Hindernissen, vielen Agenten und verstreuten Zielpositionen Konflikte auftreten,

macht die Schwierigkeit bzw. die Motivation zur Optimierung des Verhaltens aus, obwohl das Problem durch die Angabe der Distanz zur Zielposition zunächst einfach erscheint.

Tab. 4.2.: Inputreduktionstabelle für die Grundvariante des ARP.

d	Δx	Δy	x_r
W/N	<0	<0	5
E/S	<0	<0	2
N/S	<0	=0	3
W	<0	=0	4
E	<0	=0	1
N/E	<0	>0	2
W/S	<0	>0	5

d	Δx	Δy	x_r
N	=0	<0	4
W/E	=0	<0	3
S	=0	<0	1
alle	=0	=0	0
N	=0	>0	1
W/E	=0	>0	3
S	=0	>0	4

d	Δx	Δy	x_r
N/E	>0	<0	5
W/S	>0	<0	2
N/S	>0	=0	3
W	>0	=0	1
E	>0	=0	4
E/S	>0	>0	5
W/N	>0	>0	2

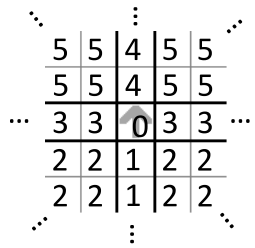

Abb. 4.7.: Aufteilung der reduzierten Inputs in Zielsegmente.

Die Struktur der Zelle (Abb. 4.8) ist durch die Verwendung von Prioritäten, einer Inputreduktionstabelle (IRT) und dynamischen Agenten sowie der erhöhten Anzahl an Attributen etwas komplexer als bei den beiden vorher vorgestellten MAS. Im Sensor werden die Kontrollen für die Multiplexer berechnet, welche bestimmen, ob für die neuen Zustände, die alten Werte der eigenen Zelle oder einer Nachbarzelle (bei Kopie eines Agenten) herangezogen werden. Die Distanzen müssen bei einer Bewegung angepasst werden. Diese Anpassung nimmt die Funktion ϕ_Δ vor. Vor der Anpassung werden die Distanzen für die IRT verwendet, genauso wie die Richtung des Agenten d. Die Prioritätenliste P wird lediglich vom Sensor gelesen aber nicht verändert. Die FSM bekommt außer dem Zustand noch den Input x_r von der IRT. Für die Löschung des Agenten muss die Funktion ϕ_τ zusätzlich zur Bewegungsbedingung m auch die Distanzen und die Bedingung für einen Swap w (wie wechseln) auswerten. Die restlichen Elemente entsprechen in Funktion und Verknüpfung der Beschreibung aus Abschn. 4.1.

Für die Berechnung der Folgezustände der Zelle werden einige Hilfsfunktionen und -variablen eingeführt. Die Positionsfunktion $pos(d, P)$ gibt an, an welcher Position eine Richtung d in der Liste P steht. Die Positionen 0-3 sind möglich, wobei 0 die höchste Priorität und 3 die niedrigste Priorität hat. Eine weitere Hilfsfunktion, durch die entschieden werden kann, welche von zwei Richtungen, die höhere Priorität hat, wird definiert:

$$h_p(z_i, d_1, d_2) = \begin{cases} 1 & \text{, falls } pos(d_1, P_i) < pos(d_2, P_i) \\ 0 & \text{, sonst.} \end{cases}$$

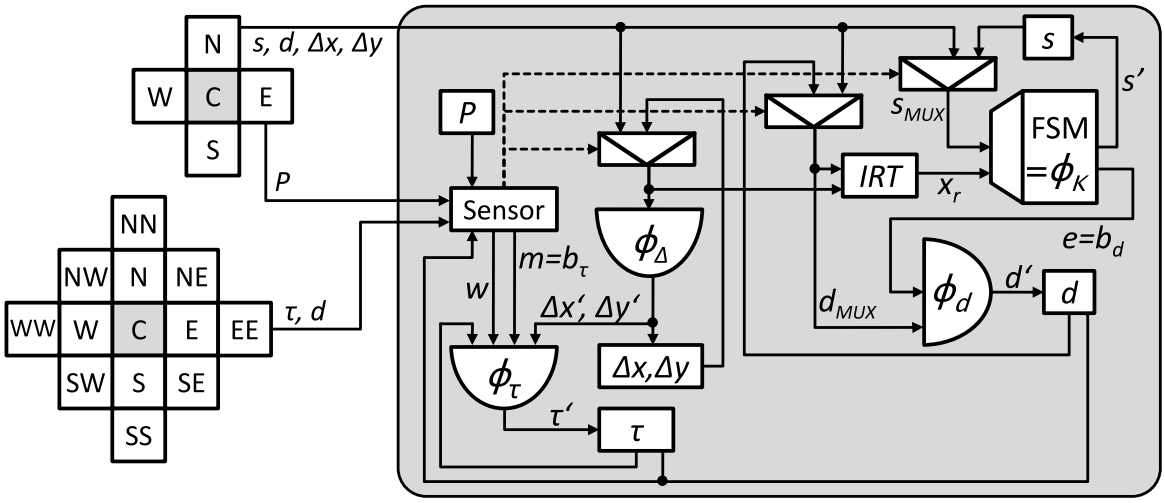

Abb. 4.8.: Zellstruktur und Nachbarschaft der Grundvariante des ARP.

Die Hilfsvariable h_{mL} und die Hilfsfunktion h werden wie in Abschn. 4.1 definiert. Die Hilfsvariable h_{mA} ist gegenüber der Variable h_{mC} wie folgt modifiziert:

$$\begin{aligned}h_{mA} = &(h(z_{NN},S)\cdot h_p(z_N,N,S)+h(z_{NE},W)\cdot h_p(z_N,E,S)+h(z_{NW},E)\cdot h_p(z_N,W,S))\cdot h(z_C,N)\\ &+(h(z_{EE},W)\cdot h_p(z_E,E,W)+h(z_{NE},S)\cdot h_p(z_E,N,W)+h(z_{SE},N)\cdot h_p(z_E,S,W))\cdot h(z_C,E)\\ &+(h(z_{SS},N)\cdot h_p(z_S,S,N)+h(z_{SE},W)\cdot h_p(z_S,E,N)+h(z_{SW},E)\cdot h_p(z_S,W,N))\cdot h(z_C,S)\\ &+(h(z_{WW},E)\cdot h_p(z_W,W,E)+h(z_{NW},S)\cdot h_p(z_W,N,E)+h(z_{SW},N)\cdot h_p(z_W,S,E))\cdot h(z_C,W).\end{aligned}$$

h_{mA} nimmt einen Wert ≥ 1 an, wenn sich ein Agent in der Zelle befindet und in dessen Blickrichtung ein Konflikt auftritt und die Priorität eines anderen beteiligten Agenten höher ist, sonst ist der Wert 0. In der Hilfsfunktion wird der Swap noch nicht berücksichtigt. Die Bedingung für einen Swap w ist

$$w = h(z_C,N)\cdot h(z_N,S)+h(z_C,E)\cdot h(z_E,W)+h(z_C,S)\cdot h(z_S,N)+h(z_C,W)\cdot h(z_W,E).$$

Die Bewegungsbedingung m ergibt sich dann wie folgt:

$$m=\begin{cases}1 & \text{, falls } (h_{mL}\geq 1\wedge \tau_C = Leer)\vee(h_{mA}=0\wedge\tau_C=Agent\wedge\tau_F=Leer)\vee w=1\\ 0 & \text{, sonst.}\end{cases}$$

Zur Bestimmung der richtigen Werte des momentanen Zustands und der momentanen Richtung über den Multiplexer wird eine weitere Hilfsfunktion eingeführt:

$$\begin{aligned}h_{MUX}(d) = &h_p(z_C,N,d)\cdot h(z_N,S)+h_p(z_C,E,d)\cdot h(z_E,W)+h_p(z_C,S,d)\cdot h(z_S,N)\\ &+h_p(z_C,W,d)\cdot h(z_W,E)+h(z_C,d_C).\end{aligned}$$

Die Hilfsfunktion h_{MUX} gibt für eine bestimmte Richtung an, ob es einen anderen Agenten aus einer anderen Richtung mit einer höheren Priorität gibt und dieser auf eine leere Zelle blickt (dann nimmt h_{MUX} den Wert 0 an). Sie wird dann für die Bestimmung von s_{MUX} und d_{MUX}, sowie für die neuen Werte der Distanzen $\Delta x'$ und $\Delta y'$ verwendet. Die Bestimmung

dieser vier Werte kann als Quadrupel bzw. transponierter Vektor aufgeschrieben werden, da die Bedingungen identisch sind, d. h. sie sind davon abhängig, ob ein Agent kopiert wird (auch beim Swap), von welcher Nachbarzelle er kopiert wird, oder ob ein Agent stehen bleibt:

$$\begin{pmatrix} s_{MUX} \\ d_{MUX} \\ \Delta x' \\ \Delta y' \end{pmatrix}^T = \begin{cases} (s_N, d_N, \Delta x_N, \Delta y_N - 1) & \text{, falls } h(z_N, S) = 1 \wedge (h_{MUX}(N) = 0 \vee h(z_C, N) = 1) \\ (s_E, d_E, \Delta x_E + 1, \Delta y_E) & \text{, falls } h(z_E, W) = 1 \wedge (h_{MUX}(E) = 0 \vee h(z_C, E) = 1) \\ (s_S, d_S, \Delta x_S, \Delta y_S + 1) & \text{, falls } h(z_S, N) = 1 \wedge (h_{MUX}(S) = 0 \vee h(z_C, S) = 1) \\ (s_W, d_W, \Delta x_W - 1, \Delta y_W) & \text{, falls } h(z_W, E) = 1 \wedge (h_{MUX}(W) = 0 \vee h(z_C, W) = 1) \\ (s_C, d_C, \Delta x_C, \Delta y_C) & \text{, sonst.} \end{cases}$$

Mit den Werten s_{MUX} und x_r kann die FSM den neuen Zustand s' sowie die Entscheidung e bestimmen. Durch die Erweiterung der Aktionsmenge des Agenten gegenüber der Grundvariante des CEP ergeben sich für die Berechnung der Aktion und der neuen Richtung folgende Formeln:

$$a = \begin{cases} Lm/Rm/Sm/Bm & \text{, falls } e = L/R/S/B \wedge m = 1 \\ Ls/Rs/Ss/Bs & \text{, falls } e = L/R/S/B \wedge m = 0 \end{cases}$$

$$d' = \begin{cases} N & \text{, falls } (d_{MUX} = E \wedge e = L) \vee (d_{MUX} = W \wedge e = R) \vee (d_{MUX} = S \wedge e = B) \\ E & \text{, falls } (d_{MUX} = S \wedge e = L) \vee (d_{MUX} = N \wedge e = R) \vee (d_{MUX} = W \wedge e = B) \\ S & \text{, falls } (d_{MUX} = W \wedge e = L) \vee (d_{MUX} = E \wedge e = R) \vee (d_{MUX} = N \wedge e = B) \\ W & \text{, falls } (d_{MUX} = N \wedge e = L) \vee (d_{MUX} = S \wedge e = R) \vee (d_{MUX} = E \wedge e = B) \\ d_{MUX} & \text{, sonst.} \end{cases}$$

Die Aktualisierung des Zelltyps ist ähnlich zur Angabe aus Abschn. 4.1, jedoch muss hier noch die Dynamik der Agenten und die Swap-Regel berücksichtigt werden, also das Löschen beim Erreichen der Zielposition und das Vertauschen der Positionen bei entsprechender Konfliktsituation, daher lautet die Formel:

$$\tau' = \begin{cases} Hindernis & \text{, falls } \tau_C = Hindernis \\ Leer & \text{, falls } (\tau_C = Agent \wedge m = 1 \wedge w = 0) \\ & \quad \vee (\tau_C = Leer \wedge ((m = 1 \wedge (\Delta x', \Delta y') = (0,0)) \vee m = 0)) \\ Agent & \text{, falls } (\tau_C = Leer \wedge m = 1 \wedge (\Delta x', \Delta y') \neq (0,0)) \\ & \quad \vee (\tau_C = Agent \wedge (m = 0 \vee w = 1)). \end{cases}$$

Die Prioritätenliste bleibt unverändert erhalten. Die Aktualisierung des gesamten Zellzustands wird dann durch die Übernahme des folgenden 6-Tupels realisiert:

$$(\tau_C, d_C, s_C, \Delta y_C, \Delta x_C, P_C) \Leftarrow (\tau', d', s', \Delta y', \Delta x', P_C).$$

Als Pseudocode formuliert (List. 4.3), wird die Zellregel wiederum etwas einfacher und kürzer als die mathematische Beschreibung, da nur Veränderungen des Zustands notiert werden. Dabei sei die Prioritätenliste in den Variablen $prN_C = pos(N, P_C)$, $prE_C = pos(E, P_C)$,

$prS_C = pos(S, P_C)$, $prW_C = pos(W, P_C)$ codiert (analog für die Nachbarzellen). Die Entscheidungen e werden mit $L = -1$, $S = 0$, $R = 1$ und $B = 2$ codiert, um eine Modulo-Rechnung für die Bestimmung der neuen Richtung zu ermöglichen.

Inputreduction$(\Delta x, \Delta y, d)$ ist eine Funktion, die aus zwei Distanzwerten und einer Richtung, den reduzierten Input für die FSM ausgibt. Wie bei den Funktionen *FSMoutput* und *FSMnewstate* wird dabei auf eine Tabelle zurückgegriffen. Diese sind im Pseudocode nicht beschrieben. Auf der obersten Ebene werden zunächst die Fälle der leeren Zelle und der Zelle mit Agent unterschieden (Z. 1 und 20). Wenn die Zelle leer ist, werden sequentiell alle Nachbarzellen geprüft und der Zustand, die Richtung und Entfernung zur Zielposition des benachbarten Agenten mit der höchsten Priorität kopiert (Z. 4 bis 10). Mit diesen Werten wird dann im Falle der erlaubten Bewegung, wenn nicht die Zielposition erreicht wurde, der neue Zustand mit den kopierten Werten berechnet und aktualisiert (Z. 13 bis 18). Wenn in der Zelle ein Agent ist, wird der Sichtbereich per Fallunterscheidung ausgewählt und zunächst auf die Swapbedingung, dann auf Konflikt und Priorität überprüft und gegebenenfalls der Agentenzustand des Nachbarn kopiert (Z. 23 bis 34). Darf keine Bewegung stattfinden, wird der Zustand aktualisert (Z. 37 bis 40). Muss ein Swap durchgeführt werden, wird der Zustand mit den kopierten Daten aktualisert (Z. 42 bis 47). Wenn der Agent sich bewegen darf, wird nur der Zelltyp aktualisiert (Z. 49).

List. 4.3: Die Zellregel des ARP als Pseudocode

```
1  if (τ_C==Leer){
2    m=0; pr=4; dx=Δx_N; dy=Δy_N; dxm = 0; dym = 0;
3    //Agent mit höchster Priorität aus Nachbarzelle kopieren (falls vorhanden):
4    if (τ_N==Agent && d_N==S){m++; s_MUX=s_N; d_MUX=d_N; pr=prN_C; dym=-1;}
5    if (τ_E==Agent && d_E==W){m++; if (prE_C<pr){
6      s_MUX=s_E; d_MUX=d_E; pr=prE_C; dx=Δx_E; dy=Δy_E; dxm=1; dym=0;}}
7    if (τ_S==Agent && d_S==N){m++; if (prS_C<pr){
8      s_MUX=s_S; d_MUX=d_S; pr=prS_C; dx=Δx_S; dy=Δy_S; dxm=0; dym=1;}}
9    if (τ_W==Agent && d_W==E){m++; if (prW_C<pr){
10     s_MUX=s_W; d_MUX=d_W; pr=prW_C; dx=Δx_W; dy=Δy_W; dxm=-1; dym=0;}}
11   //mindestens ein Bewegungswunsch und Frontzelle ungleich Zielposition?
12   if (m>=1 && ((dx+dxm!=0) || (dy+dym!=0))){
13     x_r = Inputreduction(dx, dy, d_MUX);     //Input für FSM bestimmen
14     e = FSMoutput(s_MUX, x_r);              //Aktion bestimmen
15     s_C ⇐ FSMnewstate(s_MUX, x_r);           //neuen Zustand berechnen
16     d_C ⇐ (d_MUX + e) % 4;                   //neue Agentenrichtung setzen
17     τ_C ⇐ Agent;                             //Zelltyp auf Agent setzen
18     Δx_C ⇐ dx + dxm; Δy_C ⇐ dy + dym;        //die neuen Abstände setzen
19   }
20 } else if (τ_C==Agent){
21   m = 1; w = 0; dxm = 0; dym = 0;
22   //Bewegungsmöglichkeit (inkl. Swap) überprüfen
23   if (d_C==N){
24     if (τ_N==Agent && d_N==S){w=1; s_MUX=s_N; d_MUX=d_N; dx=Δx_N; dy=Δy_N; dym=-1;}
25     else if ((d_NN==S && τ_NN==Agent && prS_N<prN_N) || (d_NE==W && τ_NE==Agent &&
         prS_N<prE_N) || (d_NW==E && τ_NW==Agent && prS_N<prW_N)||τ_N!=Leer) m = 0;
26   } else if (d_C==E){
27     if (τ_E==Agent && d_E==W){w=1; s_MUX=s_E; d_MUX=d_E; dx=Δx_E; dy=Δy_E; dxm=1;}
28     else if ((d_EE==W && τ_EE==Agent && prW_E<prE_E) || (d_NE==S && τ_NE==Agent &&
         prW_E<prN_E) || (d_SE==N && τ_SE==Agent && prW_E<prS_E)||τ_E!=Leer) m = 0;
29   } else if (d_C==S){
30     if (τ_S==Agent && d_S==N){w=1; s_MUX=s_S; d_MUX=d_S; dx=Δx_S; dy=Δy_S; dym=1;}
```

```
31      else if ((d_SS==N && τ_SS==Agent && prN_S<prS_S) || (d_SE==W && τ_SE==Agent &&
            prN_S<prE_S) || (d_SW==E && τ_SW==Agent && prN_S<prW_S)||τ_S!=Leer) m = 0;
32    } else if (d_C==W){
33      if (τ_W==Agent && d_W==E){w=1; s_MUX=s_W; d_MUX=d_W; dx=Δx_W; dy=Δy_W; dxm=-1;}
34      else if ((d_WW==E && τ_WW==Agent && prE_W<prW_W) || (d_NW==S && τ_NW==Agent &&
            prE_W<prN_W) || (d_SW==N && τ_SW==Agent && prE_W<prS_W)||τ_W!=Leer) m = 0;
35    }
36    if (m==0) { //Falls keine Bewegung
37      x_r = Inputreduction(Δx_C, Δy_C, d_C);     //Input für FSM bestimmen
38      e = FSMoutput(s_C, x_r);                  //Aktion bestimmen
39      s_C ⇐ FSMnewstate(s_C, x_r);               //neuen Zustand berechnen
40      d_C ⇐ (d_C + e) % 4;                       //neue Agentenrichtung setzen
41    } else {
42      if (w==1 && ((dx+dxm!=0) || (dy+dym!=0))){  //Falls Swap
43        x_r = Inputreduction(dx, dy, d_MUX);     //Input für FSM bestimmen
44        e = FSMoutput(s_MUX, x_r);               //Aktion bestimmen
45        s_C ⇐ FSMnewstate(s_MUX, x_r);           //neuen Zustand berechnen
46        d_C ⇐ (d_MUX + e) % 4;                   //neue Agentenrichtung setzen
47        Δx_C ⇐ dx + dxm; Δy_C ⇐ dy + dym;        //die neuen Abstände setzen
48      } else {
49        τ_C ⇐ Leer;                              //Agent löschen
50      }
51  }
```

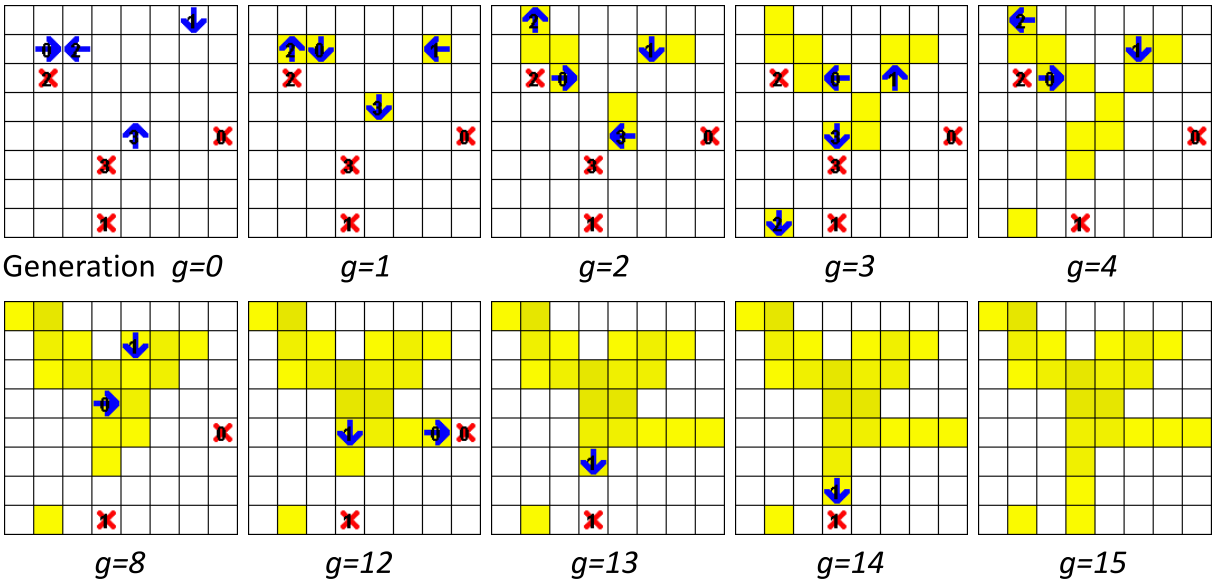

Abb. 4.9.: Sequenz einer Simulation des ARP. Die Zielpositionen der Agenten sind mit einem Kreuz markiert und mit einem Index versehen, um die Zuordnung zu sehen. Die bereits besuchten Zellen sind schattiert.

Abb. 4.9 zeigt eine Simulationssequenz des ARP auf einer initialen Konfiguration mit vier Agenten. Die Darstellung der besuchten Zellen dient nur Analysezwecken (um den Weg der Agenten nachzuverfolgen), sie sind kein für die Agenten sichtbarer Bestandteil des MAS. Die Position der Ziele der Agenten ist eine Projektion der Distanzdaten in der Zelle, in der sich der Agent gerade befindet, d. h. auf der Zielposition selbst ist für die Agenten nicht sichtbar, dass es sich um eine Zielposition handelt. Auch die verwendeten Indizes an den Agenten sind kein Teil

des MAS, sie sollen lediglich in der Abbildung eine Zuordnung ermöglichen. Die Prioritätenlisten der Zellen sowie die FSM-Zustände sind nicht dargestellt. Das Zellulare Feld hat in dieser Konfiguration keine Hindernisse und keinen Rand, d. h. die Zellen sind mit einem Wrap-Around verbunden. In der 15. Generation hat jeder Agent seine Zielposition erreicht und das globale Problem ist gelöst.

4.4 Kapitelzusammenfassung

Drei problemlösende MAS mit beweglichen Agenten wurden beispielhaft modelliert und definiert. Zu jedem Beispiel wurden die Regeln als mathematische Formeln und als Pseudocode angegeben. Die Struktur einer Zelle wurde jeweils erklärt. Beim CEP wurde zusätzlich noch ein Hardwareschaltplan einer Zelle erstellt. Das CEP, der ATAC und das ARP werden in den folgenden Kapiteln verwendet, um die Optimierungstechniken zu evaluieren.

5 Techniken zur Optimierung von Agentenverhalten mit FSMs

Zu Beginn des Kapitels wird auf die grundsätzlichen Probleme beim Entwickeln eines lokalen Agentenverhaltens zur Lösung globaler Probleme eingegangen.

Danach werden Ansätze zur Konstruktion bzw. Auffindung und Optimierung eines Agentenverhaltens beschrieben, die auf der Anpassung der Welt der Agenten beruhen. Es wird beschrieben, wann eine solche Anpassung möglich ist, ohne die Vorgaben des MAS zu verletzen und ohne das in Kap. 3 beschriebene Modell zu verlassen. Und es werden einige konkrete Ansätze zur Verhaltensoptimierung vorgestellt.

Danach werden allgemein einige heuristische Methoden und Lernverfahren aus der Literatur beschrieben und es wird dann genauer darauf eingegangen, wie diese Verfahren grundsätzlich auf FSMs und damit auch auf FSM-kontrollierte Agenten angewendet werden können. Dabei wird auch die Komplexität des Verhaltensmodells diskutiert und die Frage, ob und warum ein Einsatz von Heuristiken sinnvoll ist. Schließlich wird ein prinzipielles Verfahren für die Optimierung von FSMs entwickelt, und damit der erste Aspekt des zweiten Teilziels dieser Arbeit erreicht.

Eine weitere Möglichkeit zur Optimierung ist die manuelle Spezifizierung der Struktur der Kontrolleinheit der Agenten. Das betrifft die Fähigkeiten der Agenten, die Art der Einbettung der FSMs in die Zelle, die Größe der FSM usw. Bezogen auf die in Kap. 3 beschriebenen Beispielsysteme bedeutet das, eine andere als die Grundvariante des MAS zu verwenden. Einige grundsätzliche Techniken werden in diesem Kapitel beschrieben.

Zusätzlich zu den manuellen Techniken zur Optimierung muss die vorher entwickelte Heuristik immer eingesetzt werden, um die FSMs zu bestimmen oder zu finden. D. h. dass die manuellen Optimierungen nur als Teil der heuristischen Gesamtoptimierung zu sehen sind. Der Einsatz dieser Heuristiken kann auf viele verschiedene Arten geschehen. Es werden in diesem Kapitel Verfahren für den Einsatz von Heuristiken entwickelt, die prinzipiell auf eine Reihe von MAS in CA mit FSM-kontrollierten Agenten anwendbar sind. Die Anwendbarkeit aller heuristischen Verfahren und manuellen Techniken ist problemspezifisch und hängt auch von der gewählten Struktur des Agentenverhaltens und den Konfigurationsmöglichkeiten der Agentenwelt ab.

5.1 Grundsätzliche Fragen beim Design von lokalem Agentenverhalten

Der Entwickler eines MAS ist beim Design des Verhaltens der Agenten mit folgendem Problem konfrontiert. Er kennt das globale Problem und will ein lokales Verhalten entwickeln, das das globale Problem löst. In CA, die ein globales Verhalten basierend auf lokalen Regeln realisieren, tritt dieses Problem ebenfalls auf, auch wenn sie nicht MAS im Sinne der Modellierung aus Kap. 3 realisieren. Schulz bezeichnet das Problem allgemein (und bezogen auf CA) in [Sch81] als Lokal-Global-Problematik und stellt zwei Fragen:

„Wie müssen sich die einzelnen Individuen (lokal) verhalten, um insgesamt (global) ein bestimmtes Phänomen zu erzeugen?“ und *„Welches globale Phänomen wird durch ein bestimmtes festgelegtes lokales Verhalten erzeugt?“*

Im Kontext dieser Arbeit ist besonders die erste Frage von Bedeutung. In [BB05] wird diese Lokal-Global-Problematik ebenfalls im Zusammenhang mit CA als inverses Designproblem bezeichnet. Die einzelnen lokalen Komponenten eines Systems müssen in irgendeiner Form miteinander interagieren, um überhaupt ein globales Ziel erreichen zu können. Das bedeutet, es muss eine lokale Regel (lokales Verhalten) gefunden werden, die durch oder trotz emergenten Phänomenen, also durch Ausnutzung positiver und Vermeidung negativer Effekte der Interaktionen zwischen den Agenten (oder den einzelnen Zellen des CA), ein globales Ziel erreicht. Da Emergenz irreduzibel ist (vgl. Abschn. 2.1.4), ist es schwierig ein adäquates Verhalten zu entwickeln. Teilweise können sehr kleine Änderungen im lokalen Verhalten der Agenten schon große Auswirkungen auf das globale Verhalten des MAS haben und je komplexer das gesamte System ist, desto schwieriger wird es, das gewünschte globale Verhalten zu erreichen. In [NGB+09] heißt es dazu:

„Presently, little is known about how to design swarm intelligence systems. Thus, it is not surprising that the complexity exhibited in current implementations does not come close either to the complexity of biological systems, or to the complexity of systems that men built following the more traditional top-down approach.“

Wie man in dezentralisierten Strukturen wie einem MAS ein bestimmtes makroskopisches (globales) Verhalten erzeugt und dies auch garantieren kann, ist demnach keine leicht zu beantwortende Frage, mit der sich auch De Wolf und Holvoet in [DH05b] beschäftigt haben. Darin wird eine Methodik im Ansatz vorgeschlagen, die das makroskopische Verhalten anhand von Tests empirisch verifiziert. Anhand eines solchen Feedbacks soll dann das mikroskopische Design (das Verhalten der Agenten) angepasst werden. Über diese Anpassung wird nur gesagt, dass es generellen Richtlinien folgen soll. In Übereinstimmung mit De Wolf und Holvoet wird in [BS08] die Simulation als einzig praktikable Möglichkeit bezeichnet, ein organisches System (ein System aus autonomen interagierenden Subsystemen, das Selbst-X-Eigenschaften aufweist) mit emergentem Verhalten zu entwickeln:

„In fact, simulation seems to be the only practicable method of developing an understanding of the properties of organic systems, and we therefore conjecture that any promising design process has to involve simulation to evaluate a system's quality.“

Die Autoren beziehen sich zwar nicht explizit auf MAS, aber generell auf Systeme mit emergentem Verhalten (z. B. auch dezentralisierte MAS mit Eigenschaften eines organischen Systems). Sie empfehlen, *Evolutionäre Algorithmen* (EA) zu verwenden, um ein emergentes System zu entwickeln. Dabei spielt die Bewertung der Systeme (also des Agentenverhaltens aller Agenten) durch Simulation eine entscheidende Rolle. Die Simulation eines MAS kann sehr zeitaufwendig sein, wenn eine große Anzahl von Objekten berücksichtigt werden muss und die einzelnen Agenten nicht parallel simuliert werden können. Wenn aber eine theoretische Bewertung der Güte eines Systems nicht möglich ist, dann ist die Simulation für eine Evaluation der Qualität eines MAS notwendig. Die Bewertung eines MAS nach der Simulation kann ebenfalls schwierig sein. In vielen Fällen ist der Verlauf einer Simulation nicht deterministisch und es sind verschiedene voneinander unabhängige Ziele zu erreichen. Beim CEP z. B. sollen erstens möglichst viele

Felder besucht werden und zweitens diese möglichst schnell. Bei der Evaluierung muss die Gewichtung dieser zwei Ziele geschickt gewählt werden. Andere Möglichkeiten zur Reduzierung der Entwicklungszeit mit EA sind unter anderem die Verwendung von weniger Testumgebungen, die simuliert werden müssen oder die Abschätzung der Qualität eines Systems ohne vollständige Simulation.

Weitere Arbeiten setzen sich ebenfalls mit der Frage auseinander, wie man ein emergentes System bzw. dessen lokale Regeln optimieren kann. Anderson beschreibt in [And06] mehrere Verfahren: Zum einen ist es möglich, bekannte selbstorganisierende Systeme aus der Natur zu kopieren und gegebenenfalls anzupassen. Weiterhin gibt es einen Bottom-up-Ansatz, bei dem in einem iterativen Prozess das Verhalten nach Simulation immer wieder angepasst wird. Insbesondere für hierarchisch oder modular aufgebaute MAS und globale Ziele bietet sich auch eine Top-down-Strategie an, bei der man niedrigere Level zunächst als Blackbox ansieht und auf höherem Level ein Verhalten konstruiert, beispielsweise indem man Gruppen von Agenten zusammenfasst und ein Gruppenverhalten definiert, das zur globalen Lösung führt und später die individuellen Verhalten konstruiert, die zum gewünschten Gruppenverhalten führen. In [Edm04] wird zwischen einem konstruierenden (engl.: engineering) und einem adaptiven Ansatz unterschieden. In ersterem wird eine manuelle Konstruktion durchgeführt, in letzterem findet eine automatische Adaption statt. Beide Ansätze sind kombinierbar. In [Kom10] werden MAS sowohl manuell entwickelt als auch mit heuristischen Verfahren evolviert.

Die Anforderungen an den Entwickler eines MAS können je nach Anwendung in den Punkten Arbeitsaufwand, Entwicklungszeit und Qualität des Ergebnisses variieren. Daher können für das Design des Agentenverhaltens unterschiedliche Vorgehensweisen die jeweils besten sein. Ist z. B. die Entwicklungszeit nachrangig, könnte eine Aufzählung aller Möglichkeiten (wie in [Hal08] durchgeführt) und deren Bewertung mit empirischen Tests sinnvoll sein. Wenn eine schnelle Lösung verlangt wird und Abstriche bei der Qualität der Lösung in Kauf genommen werden können, könnte eine heuristische Methode die bessere Alternative sein. Manuelle Designs von Agentenverhalten sind in Systemen, die eine gewisse Komplexität nicht übersteigen, ebenfalls möglich. Die Verwendung eines heuristischen Verfahrens kann auch mit einer manuellen Konstruktion vom Verhalten kombiniert werden.

Auf das im letzten Kapitel entwickelte Modell eines MAS in CA bezogen, kann der Entwickler auf drei Ebenen eingreifen:

- Die oberste Ebene ist das Zellfeld. Darin kann die initiale Konfiguration verändert werden, aber auch die Topologie des Netzwerkes. Kurz: die Welt des Agenten kann angepasst werden. Dies wird in Abschn. 5.2 diskutiert.

- Die mittlere Ebene ist die Zellstruktur. Hier kann das Verhalten des Agenten angepasst werden, indem manuell die Randbedingungen für die Kontroll-FSM festgelegt werden. Kurz: Die Kontrolleinheit des Agenten kann angepasst werden, was in Abschn. 5.5 diskutiert wird.

- Die unterste Ebene ist der Inhalt der Kontrolleinheit. Diese enthält unter anderem die FSM und gegebenenfalls eine Inputreduktion und ein Aktionsmapping wie in Abschn. 3.1.2 beschrieben. Es muss also eine oder mehrere zum Konstrukt der Kontrolleinheit passende FSMs manuell oder mithilfe von Suchverfahren gefunden werden. Geeignete Verfahren werden in Abschn. 5.3 und 5.4 diskutiert.

5.2 Optimierung durch Anpassung der Welt der Agenten

5.2.1 Die Problemstellung des MAS und die Konsequenzen für die Optimierung

Man könnte die Art der Optimierung/Konstruktion in zwei Kategorien einteilen:

1. In das manuelle Konstruieren der Struktur der Kontrolleinheit und der Welt.

2. In das automatische Finden eines passenden Inhalts für die Komponenten der Struktur.

Die zweite Kategorie der Optimierung ist immer notwendig, denn ohne Inhalt der Kontrolleinheit kann kein Agentenverhalten stattfinden. Die erste Kategorie ist nur dann notwendig, wenn die Struktur oder die Welt nicht spezifiziert ist. Andernfalls kann sie sogar je nach Problemstellung unmöglich sein, da sich die Problemstellung eventuell gerade auf die gegebene Struktur bezieht. Für das Gesamtergebnis, d. h. die Güte des gefundenen Agentenverhaltens bzw. der Problemlösung ist immer die Kombination der Optimierungen auf allen Ebenen zu betrachten. Anders ausgedrückt, es müssen zwei Fragen beantwortet werden: 1. Was sollen die Agenten können? und 2. Was soll das MAS als Gesamtsystem können?

Die Problemstellung beim Optimieren ist also von entscheidender Bedeutung. Nehmen wir z. B. das CEP. Es können vier verschiedene Fälle auftreten, bei denen die Art der Optimierung erheblich variiert:

1. Es wird ein optimales Verhalten für das Ablaufen eines Grundrisses von einem ganz bestimmten Gebäude gesucht. Es gibt nur eine mögliche initiale Konfiguration.

2. Es wird ein Verhalten gesucht, das möglichst robust auf jedem beliebigen Terrain das Problem löst. Alle möglichen initialen Konfigurationen sind erlaubt.

3. Es wird ein Verhalten gesucht, das auf jedem beliebigen Terrain mit Einschränkungen in der Größe oder anderen Eigenschaften das Problem löst. Eine endliche Menge von initialen Konfigurationen ist erlaubt.

4. Es wird ein Verhalten gesucht, das auf irgendeinem Terrain das Problem löst. Die initiale Konfiguration ist dann unwichtig.

Eine Bewertung des Verhaltens der Agenten durch Simulation auf unendlich vielen Welten ist nicht möglich. Es muss im zweiten (und dritten) Fall eine Auswahl von initialen Konfigurationen gewählt werden, die möglichst repräsentativ für ein (eingeschränkt) beliebiges Terrain ist, so dass vermutet werden kann, dass ein Verhalten, das auf der Auswahl von Welten gut ist, auch auf dem Rest der Welten gut sein wird. Ein Beweis für eine solche Schlussfolgerung ist allerdings alles andere als trivial, wenn überhaupt möglich. Der vierte Fall ist beim CEP möglich, aber eher nicht sinnvoll. Beim ATAC oder ARP jedoch schon. Da würde es bedeuten, dass die Agenten sich nicht auf einem bestimmten oder beliebigen Zellfeld gut verhalten müssen, sondern dass dem Entwickler freie Hand (eventuell auch mit Einschränkungen) bei der Gestaltung der initialen Konfiguration gegeben ist. In diesem Fall müsste für die Optimierung des Gesamtsystems eine optimale Kombination von Agentenverhalten und initialer Konfiguration gefunden werden.

5.2.2 Maßnahmen zur Anpassung der Agentenwelt

Der Entwickler eines MAS kann innerhalb des in Kap. 3 beschriebenen Modells die Welt der Agenten spezifizieren (z. B. die Größe der Welt, die Topologie der Zellen, die initiale Konfiguration allgemein), wenn das Optimierungsziel nicht auf bestimmte initiale Konfigurationen festgelegt ist. Im Einzelfall kann es eine effizientere Lösung des gegebenen globalen Problems ermöglichen.

Die im Folgenden vorgestellten Maßnahmen zur Optimierung der globalen Problemlösung sind größtenteils trivial. Die Effektivität ihrer Umsetzung ist jedoch nicht trivial erkennbar. Eine Untersuchung der Effektivität findet in Kap. 6 statt. Dort werden in jedem Beispielsystem (CEP, ATAC oder ARP) unterschiedliche Problemstellungen untersucht, die zu unterschiedlicher Anwendbarkeit der Optimierungsmaßnahmen führen. Die Anwendbarkeit einer jeden der vorgestellten Maßnahmen zur Anpassung der Agentenwelt wird für jedes der drei Beispielsysteme in Abhängigkeit der Problemstellung in Abschn. 5.2.3 überprüft. Die in dieser Arbeit auf Effektivität untersuchten Maßnahmen sind:

a) Variation der Anzahl der Agenten (k) und deren initialer Positionierung

b) Verwendung von Welten mit zyklischem Wrap-Around oder mit Rand

c) Variation der Größe der Agentenwelt (Anzahl der Zellen und Seitenverhältnis)

d) Variation der Netztopologie (orthogonale oder hexagonale Struktur)

Zu a). Die Anzahl der Agenten zu erhöhen ist ein trivialer Versuch die Effektivität des MAS zu erhöhen, nach dem Motto: Je mehr Agenten „mitarbeiten“, desto schneller kann das Problem gelöst werden. Dies ist allerdings nicht zwingend der Fall, da sich Agenten auch gegenseitig behindern könnten, z. B. wenn sie als mobile Hindernisse Konflikte verursachen. In den meisten MAS ist außerdem davon auszugehen, dass die Verwendung von mehr Agenten mit mehr Kosten verbunden ist (z. B. bei Robotern). Eine unterschiedliche Positionierung der Agenten auf dem Zellfeld (auch relativ zueinander) kann ebenfalls eine große Wirkung auf das System haben, wenn es z. B. darum geht, dass die Agenten sich gegenseitig ausweichen oder treffen müssen, um zu kommunizieren.

Zu b). Die Verwendung eines Rands um das Zellfeld nimmt den Agenten einerseits die Möglichkeit, auf kurzem Weg von einer Zelle in der Nähe des Rands zu einer Zelle in der Nähe des gegenüberliegenden Rands zu kommen. Auf der anderen Seite kann der Rand als Orientierungshilfe für die Agenten dienen. Das kann in Systemen, in denen die Agenten ein bestimmtes Ziel haben oder sich treffen müssen, einen Vorteil bringen.

Zu c). Die Größe und das Seitenverhältnis der Agentenwelt beeinflussen die mittleren zu überbrückenden Distanzen zwischen Agenten und anderen Agenten oder einer möglichen Zielposition. Auf der anderen Seite beeinflussen sie auch die Wahrscheinlichkeit von Konflikten oder Kommunikationssituationen.

Zu d). Bei der Variation der Netztopologie muss beachtet werden, dass dies zwangsläufig auch eine Veränderung der Struktur der Zelle zur Folge hat, da in einer hexagonalen Topologie jede Zelle sechs direkte Nachbarn besitzt. Die Richtung der Agenten und deren Bewegungsaktionen, sowie die Konfliktbehandlung müssen entsprechend angepasst werden. Beim ARP kommt noch hinzu, dass relative Abstände zu einer Zielposition anders auszuwerten sind. Sinnvoll kann die Verwendung einer anderen Netztopologie sein, um mehr Möglichkeiten zu schaffen, sich von

einer Zelle auf eine Nachbarzelle zu bewegen. Dies kann Konflikte minimieren und Deadlocks verhindern. Eine engere Vermaschung der Zellen kann die mittleren Abstände reduzieren und gegebenenfalls Wege verkürzen.

Weitere Anpassungen sind denkbar, aber in dieser Arbeit nicht untersucht. Um die Effektivität zu untersuchen, sind zwei prinzipielle Methoden denkbar. Entweder man verwendet das gleiche Agentenverhalten in allen Welten, oder man konstruiert (oder optimiert) für jede Welt separat ein Agentenverhalten und vergleicht dann die Güte der Problemlösung. Dies sind nicht etwa zwei gleichwertige Alternativen, sondern ergeben jeweils eine andere qualitative Aussage, die bei der Präsentation der Resultate der Untersuchungen in Abschn. 6 herausgestellt wird.

5.2.3 Anwendbarkeit der Maßnahmen auf die Beispielsysteme

Die Anwendbarkeit einer Maßnahme hängt davon ab, ob in der Definition der globalen Aufgabe der Agenten bei der entsprechenden Variable ein Parameterraum zugelassen ist oder ob der Wert fix vorgegeben ist.

Anwendbarkeit der Maßnahmen auf das CEP. In der zweiten Problemstellung des CEP aus Abschn. 5.2.1 soll ein Agentenverhalten gefunden werden, das robust gegenüber Veränderungen der Anzahl und initialen Position der Agenten und gegenüber Variationen der Größe der Agentenwelt ist. Außerdem soll es für beliebig in der Welt verteilte Hindernisse (Rand mit eingeschlossen) geeignet sein. Für diese Problemstellungen sind die Maßnahmen a), b) und c) nicht anwendbar. Diese Variationen werden stattdessen verwendet, um ein in allen Welten gutes Verhalten zu entwickeln. Für die dritte Problemstellung gilt das gleiche, solange die Einschränkungen berücksichtigt werden.

In der ersten Problemstellung ist nur eine bestimmte Umgebung gegeben, in der die Agenten das CEP lösen sollen. In diesem Fall ist die Maßnahme a) zur Optimierung anwendbar, wenn man voraussetzt, dass die Ressourcen begrenzt sind (Agentenanzahl). Die Bewertung der Güte von Systemen mit unterschiedlicher Agentenanzahl ist allerdings auch nicht trivial. Ein simpler Vergleich der Anzahl der besuchten Zellen oder der Geschwindigkeit würde zu der trivialen Lösung führen, dass man einfach in alle Zellen initial einen Agenten positioniert. Realistischerweise sollte man daher für Anwendungen annehmen, dass ein Agent Kosten verursacht. Ein Vergleich der Kosten, der Geschwindigkeit und des Anteils besuchter Zellen gibt dann Aufschluss über eine geeignete Anzahl von Agenten.

Für die vierte Problemstellung kann die Welt beliebig konfiguriert werden. Es ist also möglich, die Maßnahmen b) und c) anzuwenden, um den Creatures z. B. mit einem Rand eine zusätzliche Orientierungshilfe zu geben.

Falls das CEP auf eine reale kontinuierliche Umgebung (z. B. ein Gebäudegrundriss) angewendet werden soll, steht der Entwickler vor der Frage, wie er diese Umgebung auf eine diskrete Agentenwelt aus Zellen abbildet. Die Topologie kann dann einen Einfluss auf die Qualität des entwickelten Agentenverhaltens haben. Die Anwendung von Maßnahme d) macht dann Sinn.

Anwendbarkeit der Maßnahmen auf den ATAC. Die Anzahl der Agenten zu variieren, hieße, die zu verteilende Information zu verändern. Dies kann keine sinnvolle Strategie sein, wenn eine Information gegeben ist. Variationen der Anzahl der Agenten werden also nur in dem Sinne benutzt, dass ein Verhalten entwickelt werden soll, dass für eine beliebige Anzahl von Agenten gut ist.

Angenommen, die Problemstellung ist, ein MAS zu entwickeln, das die Verteilung von Informationen unter den Agenten auf einem bestimmten Raum realisieren soll. Innerhalb dieses Raums, hat dann der Entwickler die Möglichkeit einen Rand (oder sogar Hindernisse allgemein) oder ein zyklisches Zellfeld zu verwenden. Ist der Raum nicht fest vorgegeben oder hat er nur eine obere Grenze, so kann der Entwickler auch kleinere Zellfelder benutzen. Die Maßnahmen b) und c) sind also anwendbar. Falls die Problemstellung lautet, ein Verhalten zu entwickeln, das den ATAC auf beliebig großen Feldern mit oder ohne Wrap-Around löst, dann gilt das gleiche wie für Maßnahme a). In den Arbeiten [EH08b] und [HE08] wurden Agenten für den ATAC optimiert und deren Performanz in Welten mit Rand und ohne Rand verglichen.

Für die Topologie (d)) des Netzes gilt das gleiche wie für die beiden Maßnahmen b) und c). Es ist anwendbar, wenn die Topologie nicht vorgegeben ist. Beispielsweise könnte die Anzahl der Knoten in einem Netz bekannt sein, aber die Verdrahtung soll für das ATAC optimiert werden. Dann kann eine Veränderung eine Verbesserung bewirken. Aber auch hier muss beachtet werden, dass die Zellstruktur ebenfalls modifiziert werden muss.

Anwendbarkeit der Maßnahmen auf das ARP. Beim ARP gelten die gleichen Bedingungen für die Anwendung der Optimierungsmaßnahmen wie beim ATAC. Die Anzahl der Agenten zu variieren macht nur Sinn, um ein Verhalten zu entwickeln, das für eine beliebige Anzahl gut ist. Soll das Routing auf einem bestimmten Raum geschehen, kann dieser Raum modifiziert werden (Maßnahmen b) und c)), um das MAS zu optimieren. Soll das Verhalten robust gegenüber Änderungen des Raums und der Anzahl der Agenten sein, kann das Verhalten nicht über diese Maßnahmen optimiert werden. In [EH10d] wurde eine Untersuchung über den Einfluss der Größe der Welten und dem Verhältnis der Anzahl der Agenten zur Anzahl der Zellen auf die Leistung der Agenten durchgeführt. Die Ergebnisse dieser Untersuchung werden in Abschn. 6.3.1 präsentiert.

Die Topologie des Netzes (d)) könnte von einem System, in dem Nachrichten transportiert werden sollen, unveränderlich vorgegeben sein. Dann kann man darüber keine Optimierung erzielen. Es könnte aber auch sein, dass die Entwicklung eines geeigneten Netzes mit geeigneten Start- und Zielpositionen der Agenten Teil der Optimierung des gesamten MAS ist (z. B. für einen Router auf einem Chip). In diesem Fall kann die Topologie und die Position der Agenten darin zur Optimierung variiert werden. In [EHD10] wurde für das ARP eine Topologie mit hexagonaler Struktur verwendet.

5.3 Heuristische Methoden und Lernverfahren

Eine *Heuristik* ist ein Verfahren, mit dem für ein gegebenes Problem nach einer Lösung gesucht wird, indem potenzielle Lösungen systematisch ausprobiert und bewertet werden. Nach bestimmten Schemata und unter Berücksichtigung der bisher gefundenen Lösungen und anderer Parameter (z. B. vergangene Zeit, bisher verfolgter Weg im Suchraum) werden neue Lösungen generiert oder ausgewählt und wieder ausprobiert. Oft sind dabei natürliche Entwicklungsprozesse (z. B. Evolution) das Vorbild. Diese Verfahren sind nie exakt, da sie eine zufällige Komponente besitzen. Ihr Vorteil ist jedoch, dass sie, richtig konfiguriert, bei vielen schwierigen Problemen im Durchschnitt wesentlich schneller bzw. überhaupt in akzeptabler Zeit zu akzeptablen Lösungen gelangen, als ein exaktes Verfahren. Voraussetzung für die Anwendbarkeit von Heuristiken ist also, dass nicht unbedingt die optimale Lösung gefunden werden muss, sondern andere nicht optimale Lösungen trotzdem akzeptabel sind.

Auf ein MAS angewendet, wäre die heuristisch gesuchte Lösung ein ganz bestimmtes Verhalten des Agenten. Das bedeutet, die Heuristik wird für *Maschinelles Lernen* eingesetzt. Unter dem Begriff Maschinelles Lernen werden allgemein Methoden zusammengefasst, mit denen ein Verhalten eines Programms/eines Agenten entwickelt werden kann, das sich automatisch durch die Verarbeitung von Eingaben/Sensordaten anpassen kann und mit der Zeit verbessert. Umgangssprachlich: Das Programm/der Agent lernt aus seiner Erfahrung. In dieser Arbeit werden FSMs gesucht, die ein Agentenverhalten induzieren, welches das Problem optimal oder akzeptabel löst. Was genau akzeptabel (und auch optimal) ist, muss im jeweiligen Fall definiert werden.

Man macht eine generelle Einteilung der Lernverfahren in *Überwachtes Lernen*, *Bestärkendes Lernen* und *Unüberwachtes Lernen* (z. B. in [RN04, S. 794f.]). Beim überwachten Lernen soll eine von vorneherein bekannte Funktion realisiert werden. Das bedeutet, zu jedem möglichen Input oder jeder möglichen Folge von Inputs gibt es einen bekannten gewünschten Output oder eine Folge von Outputs. Wenn es sich beim lernenden Element um einen Agenten handelt, heißt das, dass sein individuelles Verhalten bekannt ist, nur dessen Realisierung noch angelernt werden muss. Zu jedem möglichen Input oder jeder möglichen Folge von Inputs über die Sensorik des Agenten, gibt es eine konkrete Aktion oder eine Folge von Aktionen, die der Agent ausführen soll. Eine übergeordnete Einheit (der Lehrer oder Trainer) überwacht die Aktionen des Agenten und vergleicht sie mit den gewünschten Aktionen. Als Rückkopplung für den Agenten wird die exakte Abweichung zum gewünschten Output zur Korrektur der Entscheidung verwendet. Durch Anpassung des Verhaltens kann dann die gewünschte Funktion ganz oder annähernd realisiert werden. Diese Form des Lernens macht für die Optimierung des Agentenverhaltens im Kontext der hier behandelten Problemstellung keinen Sinn, denn wenn die gewünschte Funktion bekannt wäre, bräuchte man sie nicht mehr zu suchen. Bezogen auf das komplette MAS kann man aber Verfahren des überwachten Lernens einsetzen. Dann ist die bekannte und zu realisierende Funktion das gewünschte globale Verhalten, das durch das gesuchte Agentenverhalten realisiert werden soll. Alternativ könnten lokale, kurzfristige Ziele für die Agenten definiert werden, mit denen es möglich ist, nicht die Aktion, sondern deren Effekt zu bewerten. Diese lokalen Ziele müssten dann allerdings so geschickt gewählt werden, dass sie auch zum globalen Ziel führen, was in emergenten Systemen nicht trivial ist.

Beim *Bestärkenden Lernen* erhält das lernende Element ebenfalls ein Feedback. Aber es wird nur mit Belohnungen und Strafen als Rückkopplung gearbeitet. Die Belohnungen und Strafen ergeben sich als direkte Folge aus den Entscheidungen des Lerners. Eine Funktion, die das gewünschte Verhalten repräsentiert ist nicht bekannt, d. h. das Feedback ist ohne exakte Bestimmung eines Fehlers. Im Fall der Agenten, wird dieses Feedback aus der Umwelt wahrgenommen und intern (im Agenten) als wünschenswert oder nicht wünschenswert eingestuft und daraufhin das Verhalten angepasst. Um solch ein Verfahren für globale Probleme sinnvoll einzusetzen, müssen die Feedbacks auf die lokalen Aktionen entsprechend gewählt werden. Dies ist das gleiche nichttriviale Problem wie beim überwachten Lernen mit lokalen Teilzielen.

Beim unüberwachten Lernen sind die Outputs der zu lernenden Funktion nicht im Voraus bekannt. Auf den Agenten bezogen heißt das, dass dessen Aktionen nicht in Abhängigkeit der Inputs vorher definiert sind. Stattdessen soll eine vom Ziel unabhängige Repräsentation der Inputs erlernt werden. Dadurch können Vorteile erworben werden, wie z. B. die Minimierung redundanter Informationen oder eine Klassifizierung mit Verlust unwichtiger Informationen. Damit hat ein Agent aber immer noch kein Verhalten erlernt. In [Sch93, S. 84] heißt es:

„Unüberwachtes Lernen für sich allein macht keinen Sinn. Es ist dann berechtigt, wenn es den Lernvorgang erleichternde 'Präprozessoren' für die Eingaben zielgerichteter Systeme ermöglicht."

Das bedeutet, unüberwachtes Lernen kann auf die Agenten nur in Kombination mit einer überwachten Lernmethode angewendet werden.

Im Folgenden wird das unüberwachte Lernen nicht weiter betrachtet. Zwei prinzipielle Möglichkeiten zur Optimierung eines Verhaltens durch maschinelles Lernen sind gegeben:

- **Verhaltensoptimierung mit lernenden Agenten** (Abb. 5.1(a)): In Abschn. 2.1.1 wurden lernende Agenten beschrieben, die ihr Verhalten anpassen können, während sie im MAS ihrer Aufgabe nachgehen. In diesem Fall ist für die Qualität des globalen Verhaltens des MAS nicht nur das erlernte Verhalten, das die Agenten am Schluss (wenn das Problem gelöst ist oder eine Abbruchbedingung eintritt) besitzen, von Bedeutung, sondern alle Verhaltensweisen von Beginn an. Das Verhalten des Agenten ist hier nicht als Ergebnis aufzufassen, sondern vielmehr ist der gesamte Lernprozess als Ergebnis anzusehen. Alternativ kann man, nachdem man die Agenten auf diese Weise angelernt hat, das Verhalten, das die Agenten am Schluss besitzen, in nicht lernenden Agenten gleich zu Beginn der Simulation des MAS verwenden. Dies führt zu einer neuen Bewertung des Verhaltens, das nicht mehr weiter vom Lernverfahren abhängig ist.
- **Verhaltensoptimierung mit nicht lernenden Agenten** (Abb. 5.1(b)): Ein grundsätzlich anderer Ansatz, das Verhalten eines Agenten anzupassen ist es, nicht lernende Agenten zu verwenden, also solche, die ein während einer Simulation konstantes Verhalten haben. Am Ende einer Simulation ist dann die Qualität, d. h. die Güte der globalen Lösung, zu bewerten und danach ein durch ein bestimmtes Verfahren verändertes Verhalten wieder zu simulieren. Bei diesem Verfahren können bisherige Simulationsergebnisse berücksichtigt werden. So erhält man schrittweise eine immer bessere globale Lösung bzw. ein besseres Agentenverhalten. Es lernt aber dennoch nicht der einzelne Agent, sondern das gesamte System.

Welche der beiden Möglichkeiten mit welchem Lernverfahren zum Einsatz kommen kann, hängt stark mit der Modellierung des Agentenverhaltens zusammen. Nicht jedes Verfahren ist z. B. auf FSMs anwendbar. In einigen Fällen sind die Strukturen des Verhaltensmodells geradezu prädestiniert für bestimmte Lernverfahren, da sie gerade aus diesem Grund entwickelt worden sind. Bereits erwähnt wurden die Zusammenhänge des Modells und der Lernverfahren bei den Künstlichen Neuronalen Netzen und den Classifier-Systemen (Abschn. 3.2.2).

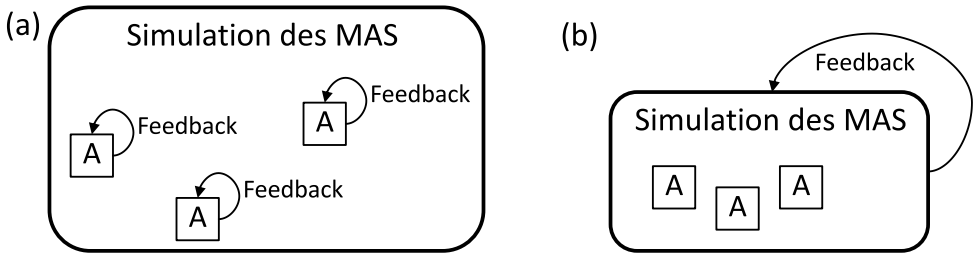

Abb. 5.1.: Verhaltensoptimierung mit (a) lernenden und (b) nicht lernenden Agenten mit Rückkopplung.

In diesem Abschnitt werden exemplarisch heuristische Verfahren und Lernmethoden kurz vorgestellt: *Genetische Programmierung*, *Genetische Algorithmen* und *Q-Learning*. Es gibt sehr viele

weitere Verfahren mit jeweils unterschiedlichen Varianten und Anwendungsgebieten, so dass die hier vorgestellten Verfahren keine komplette Liste darstellen können. Es soll vielmehr mit Blick auf MAS im Allgemeinen und FSMs im Speziellen eine Idee vermittelt werden, welche Optimierungsverfahren typischerweise angewendet werden können.

5.3.1 Motivation für die Anwendung heuristischer Verfahren

Zunächst stellt sich die Frage, warum überhaupt Optimierungsverfahren, die nicht exakt sind, auf FSM-kontrollierte Agenten angewendet werden sollten. Die Antwort darauf hat mit dem Suchraum aller Möglichkeiten, eine FSM mit beschränkter Größe zu codieren (die Inhalte der Zustandsübergangstabelle festlegen) zu tun. Angenommen, die Anzahl der möglichen Zustände s der FSM sei beschränkt auf eine bestimmte Zahl $\#s = n$. Sei weiterhin die Anzahl der möglichen Inputs x beschränkt auf $\#x = i$ und die Anzahl der möglichen Outputs y beschränkt auf $\#y = o$. Dann gibt es

$$|K| = (n \cdot o)^{(n \cdot i)}$$

Möglichkeiten, die Zustandsübergangstabelle zu codieren. Es gibt also $|K|$ *Kandidaten*, die eine mögliche Lösung für das Optimierungsproblem darstellen. K ist dabei die Menge aller Kandidaten. An der Formel ist zu erkennen, dass diese Menge mit der Anzahl der Zustände und der Anzahl der Inputs exponentiell wächst. Mit kleinen Werten n, o und i sowie einer Simulation, die in angemessener Zeit durchgeführt werden kann, kann die Qualität jeder Lösung durch Aufzählen aller Möglichkeiten (und Simulieren) festgestellt und verglichen werden. Zu beachten ist hierbei, dass alle FSMs, die mit weniger Zuständen codiert werden können, in der Menge K enthalten sind. Es handelt sich dabei um FSMs, die eine Zustandsreduktion erlauben oder unerreichbare Zustände besitzen. Des Weiteren ist zu beachten, dass nicht alle unterschiedlich codierten FSMs auch ein unterschiedliches Verhalten des Agenten erzeugen. In diesem Sinne äquivalente FSMs können durch Zustandspermutationen oder die eben genannten Reduktionen erzeugt werden. Unter der Annahme, dass es einen festen Startzustand $s = 0$ gibt, kann man $(n-1)!$ Permutationen der restlichen Zustände erzeugen, die ein identisches Verhalten codieren.

Diese Äquivalenzen erlauben eine Reduzierung der relevanten Menge an potentiellen Lösungen. In [Hal08] wurde ein spezielles Aufzählungsverfahren und spezielle Hardware zum Simulieren verwendet, um für das CEP mit 26 initialen Konfigurationen (alle mit $k = 1$ Agent und zwischen 56 und 684 Zellen, Abb. B.1 und B.2) alle relevanten 6-Zustands-FSMs zu testen. Die Anzahl der Inputs und der Outputs sind bei der Grundvariante des CEP $i = o = 2$. Der Suchraum hat dann eine Größe von $|K| = 12^{12} = 8.916.100.448.256$. Durch das Aufzählungsverfahren konnte der Suchraum auf 14.762.149.668 Kandidaten reduziert werden.

Die Simulationszeit betrug über drei Monate, trotz der mit $n = 6$ geringen Anzahl an Zuständen, einer Reduzierung der Kandidaten um zwei bis drei Größenordnungen und einer strengen Abbruchbedingung bei zu schlechtem Verhalten auf den ersten 5 von 26 Welten. Agenten, die mehr Inputs verarbeiten, eine größere Aktionsmenge haben oder mehr Zustände besitzen, können mit diesem Verfahren nicht mehr exakt optimiert werden. Natürlich gilt das dann auch für FSMs mit nicht beschränkter Anzahl von Inputs, Outputs und Zuständen.

Unter der Annahme, dass man das Agentenverhalten im Sinne des globalen Ziels des MAS verbessern kann, indem man den Agenten mehr Aktionen zur Verfügung stellt, ihnen mehr Informationen aus ihrer Umwelt bereitstellt und ihnen mehr Kapazitäten zum Speichern von

internen Zuständen gibt, muss man schlussfolgern, dass Aufzählungstechniken nicht mehr ausreichen um ein gutes Verhalten zu entwickeln.

Die Entwicklung eines exakten Algorithmus zur Suche innerhalb der relevanten Menge von FSMs gestaltet sich ebenfalls schwierig, weil von der Codierung einer FSM nur durch Simulation auf die Güte des Verhaltens geschlossen werden kann und weil die Auswirkungen von Änderungen in der Codierung ebenfalls nur durch Simulation erfasst werden können. Das bedeutet, es ist nicht klar, wie eine gute FSM grundsätzlich gestaltet werden muss, und wie eine bereits gefundene FSM verändert werden muss, um ein verbessertes Verhalten zu erzeugen. Außerdem ist das Ziel, ein Verfahren zu entwerfen, das für beliebige MAS mit FSM-kontrollierten Agenten eingesetzt werden kann. Das macht eine systematische Gestaltung einer FSM unmöglich, da nicht bekannt ist, welche Outputs und welche Inputs es gibt. Ein allgemeines Verfahren für eine systematische Annäherung (gegebenenfalls mit exaktem Endergebnis) an die optimale FSM in einem angemessenen Zeitrahmen scheint daher nur als zufallsbehaftetes heuristisches Verfahren realisierbar zu sein.

5.3.2 Evolutionäre Algorithmen

Evolutionäre Algorithmen (EA) sind heuristische Optimierungsverfahren, die bei der Suche nach einer Lösung die natürliche Evolution nachahmen. Eine Einführung in EA und verschiedene Varianten von EA findet man unter anderem in [ES03]. Die prinzipielle Funktionsweise einiger Varianten wird im Folgenden beschrieben. Allen EA gemeinsam ist das grundlegende Prinzip der Verbesserung von Lösungen bzw. Lösungskandidaten (auch *Individuen*) durch iterative Wiederholung einer „natürlichen Selektion“ nach dem Prinzip des „Survival of the fittest“. Jedem Lösungskandidaten kann mit einer Qualitätsfunktion eine *Fitness* f zugeordnet werden, die vergleichende Aussagen über die Güte der Lösung erlaubt. Der EA arbeitet auf einer Liste von möglichen Lösungskandidaten, der *Population* P. Zu Beginn muss diese Population mit (zufälligen) Lösungskandidaten initialisiert und deren Fitness bestimmt werden. Danach finden wiederholt folgende Schritte statt, bis entweder eine Grenze der Berechnungszeit oder eine bestimmte Fitness erreicht wurde:

1. *Selektion* von *Eltern*. Aus der Population wird eine Untermenge ausgewählt, aus denen neue Lösungen generiert werden sollen. Die Selektion kann zufallsbehaftet sein und die Fitness der Kandidaten berücksichtigen.

2. *Rekombination*. Aus den als Eltern selektierten Kandidaten werden durch eine bestimmte Technik neue Kandidaten generiert. Dafür ist eine geeignete Codierung der Kandidaten notwendig. Die Codierung wird in diesem Zusammenhang auch als *Genom* oder *Chromosom* bezeichnet. Die Rekombination erstellt aus zwei (oder auch mehr) Elterngenomen neue Genome, indem Teile des Genoms aus allen Elternteilen übernommen werden. So wird die Menge der *Nachkommenschaft* O (engl.: *Offspring*) erzeugt.

3. *Mutation*. Die Genome werden an bestimmten Stellen zufällig verändert. Mutation und Rekombination können auch kombiniert in beliebiger Reihenfolge verwendet werden, d. h. die Mutation kann an den Eltern oder an den Nachkommen durchgeführt werden.

4. *Evaluierung* und *Substitution* für die nächste Iteration[1]. Der Fitnesswert der Nachkommen wird bestimmt und mittels Vergleich mit den Fitnesswerten der alten Population eine neue Population erstellt, in der die „fittesten überleben“.

Je nach Art der Repräsentation der Kandidaten, also der Codierung ihres Genoms, nach Art der Rekombination und Mutation sowie nach Selektionsmechanismus unterscheidet man verschiedene EA-Varianten, von denen im Folgenden zwei beschrieben werden, Genetische Programmierung und Genetische Algorithmen.

Genetische Programmierung. Genetische Programmierung (GP) geht zurück auf John Koza [Koz92]. Die Individuen der Population werden hier mit Baumstrukturen repräsentiert und können als Programme angesehen werden. Die Programme bestehen aus elementaren Operationen aus einer endlichen Menge von elementaren Operationen und sind variabel in ihrer Länge. Die Mutation und Rekombination basiert auf dem Austausch dieser elementaren Operationen oder von Folgen solcher Operationen. In der Baumstruktur heißt das, dass Teilbäume (gegebenenfalls unterschiedlicher Länge) ausgetauscht werden.

Koza beschreibt in [Koz94] einige Beispiele von programmgesteuerten Agentensystemen, allerdings mit jeweils nur einem Agenten. Dem CEP sehr nahe kommt das *Obstacle-Avoiding Robot*-System ([Koz94, Kap. 13]). Abgesehen von der Anzahl der Agenten ist der Unterschied zum CEP, dass der Agent sich nur nach links drehen kann, dafür aber mehrere Zellen überspringen kann. Das Verhalten des Agenten wird durch die sequentielle Ausführung der elementaren Operationen des Baumes festgelegt, wobei es auch bedingte Operationen gibt. Die Aktionen des Agenten entsprechen dabei einer Teilmenge der elementaren Operationen. Es gibt auch Operationen, die lediglich einen Rückgabewert liefern, der dann als Parameter für eine Aktion verwendet werden kann. Übertragen auf ein MAS im CA-Modell führt der Agent pro Generation nur eine elementare Operation seines Programms aus. Ein ähnliches System, in dem aber mehrere Agenten programmgesteuert navigieren, um einen Ausgang zu finden, wird in [BMM+08] durch GP optimiert.

Für die Anwendung von GP auf FSM-kontrollierte Agenten muss jedoch ein anderer Ansatz als die Ausführung einer elementaren Operation pro Generation gewählt werden. In [KEFH09] wird dafür ein Ansatz vorgestellt. Darin wird die Zustandsübergangstabelle als Baumstruktur dargestellt und bei der Rekombination und der Mutation bestimmte Teile des Baums immer unverändert belassen, um die Struktur wieder auf ein Äquivalent zur FSM zurückführen zu können. Für die Grundvariante des CEP und eine Beschränkung der FSM auf $n = 6$ Zustände werden die folgenden Operationen benötigt: $Lm, Rm, Ls, Rs, if(s = 0), if(s = 1), \ldots, if(s = 5), if(m = 1), s' = 0, \ldots, s' = 5$. Die Programmstruktur ist in Abb. 5.2 angegeben. Operationen, die den Aktionen des Agenten entsprechen, können im Laufe der Evolvierung nur durch andere elementare Operationen, die Aktionen sind, ersetzt werden. Gleiches gilt für Operationen, die einen neuen Zustand setzen. Im Unterschied zu den Agentenprogrammen von Koza wird hier in jeder Generation das ganze Programm ausgeführt.

Durch die Beschränkung der Anzahl der Zustände wird auch die Länge des genetischen Programms festgelegt, was einen Vorteil des GP gegenüber anderen EA mit Genomen fixer Länge eliminiert. Außerdem ist die Codierung einer FSM als ein solches Programm eher umständlich, was sich auch in der Unveränderlichkeit einiger elementarer Operationen ausdrückt. Als Optimierungsverfahren für FSM-kontrollierte Agenten eignet sich GP also nur bedingt.

[1] Die Population jeder Iteration wird in der Literatur auch Generation genannt. Um Verwechslungen mit dem Generationsbegriff in Zellularen Automaten zu vermeiden wird hier nur von Iteration gesprochen.

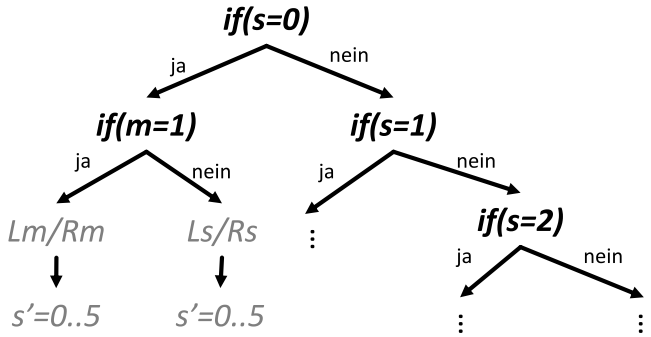

Abb. 5.2.: Struktur eines genetischen Programms zur Agentensteuerung von FSM-kontrollierten Agenten nach [KEFH09]. Fett dargestellte Operationen können nicht verändert werden.

Genetische Algorithmen. Genetische Algorithmen (GA) wurden von John Holland erfunden [Mit96, S. 2]. In GA werden die Individuen einer Population nicht als Baumstruktur oder Programm codiert, sondern als String von Bits, Zahlen oder Zeichen. Im Standardfall ist die Länge des Strings von vorneherein festgelegt ([Koz94, S. 21f.]). Je nach Semantik des Genoms ist das zwar nicht unbedingt notwendig, erleichtert aber die Mutations- und Rekombinationsverfahren, bei denen Teile der Strings ausgetauscht werden. Über absolute Positionsangaben kann das nur bei Strings gleicher Länge passieren.

Im Bereich des Künstlichen Lebens wurden GA eingesetzt, um das Verhalten von Agenten, die durch eine FSM gesteuert sind, zu evolvieren. In [RDHK99] werden Mealy-FSMs für die Kontrolle von virtuellen Kreaturen eingesetzt und mit GA optimiert[2].

In [JCC+90] müssen Agenten (dort künstliche „Ameisen“) auf einem zweidimensionalen Gitter einen Pfad möglichst akkurat ablaufen. Es befindet sich dabei nur ein einzelner Agent in dem System. Gesteuert wird er ebenfalls von einer Mealy-FSM mit maximal 32 Zuständen einem binären Input und 4 möglichen Aktionen als Output. Die Mealy-FSM wird binär codiert, indem die Inhalte der Zustandsübergangstabelle (5 Bit um den Anfangszustand zu spezifizieren; 7 Bit für den Folgezustand und den Output für jede der 64 Kombinationen von Input und aktuellem Zustand) konkateniert werden. Eine FSM hat damit 453 Bits. Die Rekombination kann auf diesem Bitstring erfolgen, indem bitweise der Inhalt vertauscht wird. Eine Mutation geschieht, indem an einzelnen Positionen des Strings das Bit invertiert wird. Die Bewertung eines Individuums erfolgt über eine Simulation des Systems.

In [Mar04] werden Mealy-FSM-kontrollierte Agenten eingesetzt, um die Futtersuche von Ameisen in zweidimensionalen Gittern zu simulieren. Die FSMs mit maximal 4 Zuständen, 4 möglichen Inputwerten und 6 möglichen Outputs werden mit GA evolviert. Die Codierung des Genoms erfolgt nicht binär, sondern als ein String von konkatenierten Transitionen. Jede Transition besteht aus 4 Werten, dem aktuellen Zustand, dem Folgezustand, dem Input und der Aktion. In dieser Codierung sind die Kombinationen von Input und aktuellem Zustand explizit angegeben und nicht wie im vorigen Beispiel implizit durch die Position im Bitstring gegeben. Es können Genome existieren, in denen einige Kombinationen nicht enthalten sind, d. h. in der FSM fehlen diese Transitionen, was bei entsprechendem Input zur Aktionslosigkeit führt.

[2] Für Details zur Rekombinations- und Mutationstechnik wird auf eine nicht mehr existente Internetquelle verwiesen.

Die in dieser Arbeit gewählte Repräsentation der FSMs als Genom ist anders. Sie basiert nicht auf einer binären Codierung der Inhalte der Zustandsübergangstabelle, sondern fasst semantisch zusammenhängende Teile als unteilbare Elemente des Strings auf und entspricht der Repräsentation einer FSM in [SG00b, SG00a]. Das bedeutet konkret, ein Paar aus Folgezustand s' und Ausgabe e ist ein nicht teilbarer Bestandteil des Genoms (Abb. 5.3). Diese Teile können in Analogie zur Biologie auch *Gene* genannt werden. Die Idee hinter dieser Aufteilung ist, die Semantik eines Zustandsübergangs bei der Rekombination zu erhalten. Angenommen, zu einem fortgeschrittenen Zeitpunkt der Evolution gibt es bei der Rekombination in einer der beiden Eltern-FSMs vom Zustand $s = 0$ einen Übergang zu $s' = 1$ und in der anderen einen Übergang zu $s' = 2$. Da beide Eltern noch in der Population sind, müssen sie gute Eigenschaften haben, die sich möglicherweise in einem der beiden Übergänge manifestieren. Also wird es sinnvoll sein, wenn der Nachkomme ebenfalls einen der beiden Übergänge bekommt. Würde man stattdessen mit der binären Codierung arbeiten, könnte es vorkommen, dass weder der eine noch der andere Übergang in den Nachkommen übernommen wird (aus 001 und 010 kann z. B. 011 werden). Der Startzustand ist hier implizit der Zustand 0, so dass diese Angabe im Genom wegfallen kann. In [SG00a] werden die auf diese Weise codierten Mealy-FSMs eingesetzt, um einen Agenten in einem Spiel auf einem zweidimensionalen Gitter zu steuern. Durch einige Erweiterungen in der Codierung ist sogar eine Evolution von FSMs mit beschränkter aber variabler Anzahl von Zuständen möglich.

$3/e_0$	$0/e_0$	$1/e_0$	$4/e_0$	$5/e_1$	$2/e_1$	$1/e_0$	$3/e_1$	$5/e_0$	$0/e_1$	$2/e_0$	$4/e_0$

Abb. 5.3.: Beispiel einer als Genom für einen GA codierten FSM. Die FSM entspricht der 6-Zustands-FSM aus Tab. 3.1. Das Genom besteht aus 12 Genen (nicht teilbaren Paaren (s'/e)).

Die Codierung eines FSM-kontrollierten Agenten als String ist einfach, aber sie muss auch für das jeweilige Problem geeignet sein. Eine erfolgreiche Suche basiert unter anderem darauf, dass ähnliche Codierungen von FSMs auch ähnlich gute Fitnesswerte haben. Das bedeutet, dass eine leichte Änderung im Genom im Generellen auch nur eine leichte Änderung der Fitness hervorrufen soll. Für die Funktion eines GA ist eine geeignete Codierung daher von entscheidender Bedeutung, weil sie dem Suchraum eine bestimmte Struktur gibt, d. h. sie definiert eine Nachbarschaftsfunktion, nach der alle FSMs, die sich nur an einer Stelle im Genom unterscheiden benachbart sind. Der Suchraum muss einigermaßen konsistent sein, um überhaupt lokale und globale Maxima ausmachen und sich diesen annähern zu können. In [KEFH09] wurde am Beispiel des CEP und FSMs mit 6 Zuständen durch Überprüfung der Fitnesswerte von benachbarten Lösungen gezeigt, dass der Suchraum zwar relativ inkonsistent ist, aber dennoch einige benachbarte Lösungen ähnliche Fitnesswerte haben, so dass die Anwendung eines EA (GA oder GP) erfolgversprechend erscheint.

Die Literatur zeigt, dass es prinzipiell möglich ist, GA für die Optimierung einzusetzen. In [Pet06] werden noch weitere Beispiele für die Evolvierung von FSMs mit EA aufgezählt. Im Rahmen einer Untersuchung zur prinzipiellen Anwendbarkeit von GA auf FSM-kontrollierte Agenten wird in Abschn. 5.4 ein GA mit detaillierten Angaben zu Selektion, Mutation, Rekombination und Substitution entwickelt.

5.3.3 Q-Learning

Ein zur Kategorie des bestärkenden Lernens gehörendes Verfahren ist das *Q-Learning*. Eine Einführung in Q-Learning findet man unter anderem in [Mit97, Kap. 13]. Die Funktionsweise des Q-Learning basiert auf der Zuordnung von Werten zu allen Entscheidungen, die ein Agent treffen kann, in Abhängigkeit seines aktuellen Zustands. Das bedeutet, jeder Kombination aus Zustand und Entscheidung wird ein sogenannter *Q-Wert* zugeordnet und am Anfang mit 0 initialisiert[3]. Diese Q-Werte werden vom Agenten herangezogen, wenn er sich entscheiden muss und im Verlauf der Simulation des MAS aufgrund von Rückmeldungen der Umgebung (positiv wie negativ) verändert. Eine allgemeine Formel zur Aktualisierung der Q-Werte nach einer Entscheidung ist nach [Mit97, S. 382]:

$$Q(s_t, e_t) \leftarrow (1-\alpha)Q(s_t, e_t) + \alpha[r + \max_e(Q(s_{t+1}, e))].$$

Der neue Q-Wert für die aktuelle Entscheidung e_t in dem aktuellen Zustand s_t ergibt sich aus einer durch den Lernfaktor α festgelegten Gewichtung aus dem alten Wert und dem um die Belohnung r erhöhten maximalen Q-Wert (Maximum über alle möglichen Entscheidungen) in der nächsten Generation. Dies ist nur eine allgemeine Formel, die im speziellen Fall angepasst werden kann, z. B. mit weiteren Gewichtungsfaktoren. Um das Lernen effektiv zu gestalten, muss die Belohnungsfunktion r der Zielsetzung des Agenten oder des MAS entsprechend angepasst sein. Der Wert, den r annimmt, muss nicht positiv sein. Bei negativen Werten kann man auch von einer Bestrafung sprechen. Der Agent wählt in jedem Schritt entweder komplett zufällig oder teilweise zufällig in Abhängigkeit der Q-Werte eine der Entscheidungen aus und generiert damit einen neuen Q-Wert für die Kombination aus Zustand und Entscheidung. Mit der Zeit sollten die Q-Werte für erfolgreiche Entscheidungen (solche mit hohen Belohnungen) mit größerer Wahrscheinlichkeit ausgewählt werden.

In [OL09] wurde das CEP mit erweiterter Aktionsmenge mit Q-Learning optimiert und dabei die Belohnungsfunktion so gestaltet, dass eine positive Belohnung ausgegeben wurde, wenn eine bis dato noch nicht besuchte Zelle besucht wurde. Die Autoren haben ebenfalls eine Methode entwickelt, mit der man über Q-Learning zu Mealy-FSM-kontrollierten Agenten kommt. Dabei wird nicht nur für jedes Paar aus Zustand und Aktion, sondern für jede Kombination aus Eingabewerten mit Zustand und Aktion mit (Folge-)Zustand ein Q-Wert vergeben. Insgesamt gibt es dann $(n \cdot i) \cdot (n \cdot o)$ Q-Werte. Nach einer vorher festgelegten Zeit des Lernens, kann für jedes Paar (s, x) das Paar (s', y) mit dem bis dahin höchsten Q-Wert ausgewählt werden, um in eine Zustandsübergangstabelle eingetragen zu werden.

Durch die mit den Ein- und Ausgaben sowie der Anzahl der Zustände wachsenden Tabelle der Q-Werte benötigt Q-Learning für die Simulation mehr Ressourcen als die EA-Verfahren. Für massivparallele Modelle wie den CA scheint das Verfahren daher nur bedingt geeignet.

5.3.4 Weitere Optimierungsmethoden

Es gibt eine große Zahl von Optimierungsmethoden und Varianten, die prinzipiell auf FSMs anwendbar sein könnten. Die hier erwähnten Methoden können aufgrund der Vielfalt nicht

[3] In der Literatur wird meist nicht von Entscheidungen, sondern Aktionen gesprochen, da anders als hier davon ausgegangen wird, dass Entscheidungen und Aktionen äquivalent sind.

vollständig sein und nicht detailliert beschrieben werden. Es sollen jedoch noch einige alternative Möglichkeiten zur heuristischen Optimierung von FSMs betrachtet werden, die teilweise auch als Ergänzung zu anderen Heuristiken eingesetzt werden können.

Ant Colony Optimization. Ein von der Futtersuche der Ameisen inspiriertes heuristisches Verfahren ist die Ant Colony Optimization (ACO) [DD99]. Bei der Futtersuche setzen Ameisen Pheromone ein, die sie auf der zurückgelegten Strecke ablegen. Sie nehmen die Pheromonkonzentration auf dem Weg wahr und werden dadurch in ihrer Wegwahl beeinflusst. Sie bevorzugen, wenn es Alternativen gibt, mit einer bestimmten Wahrscheinlichkeit den Weg mit der höheren Pheromonkonzentration, also den Weg, auf dem schon viele andere Ameisen Pheromone abgesetzt haben und sich diese noch nicht verflüchtigt haben. Angenommen, es gebe vom Nest zu einer Futterquelle einen langen und einen kurzen Weg, so würden zu Beginn beide Wege mit gleicher Wahrscheinlichkeit ausgewählt werden, aber die Intensität der Pheromone auf dem kürzeren Weg schneller steigen, da hier die Ameisen nach kürzerer Zeit wieder zurückkehren. Die folgenden Ameisen werden nun bevorzugt den kürzeren Weg benutzen und damit die Wirkung immer weiter verstärken.

Die Suche nach dem kürzeren (kürzesten) Weg kann auf ein Optimierungsproblem, also auf die Suche nach der besseren (besten) Lösung, übertragen werden. In einem iterativen Prozess werden von virtuellen Agenten (den Ameisen) wiederholt Lösungen generiert, die einem möglichen Weg zur Futterquelle entsprechen. Alle möglichen Lösungen müssen dabei auf einen möglichen Weg zum Futter abgebildet werden. Eine Idee zur Abbildung der Wege auf FSMs ist die Verwendung einer Matrix, in der für jede Kombination (s, x) eine Spalte und für jede Kombination (s', y) eine Zeile existiert. Das sind die gleichen Einträge wie in der Q-Wert-Tabelle beim Q-Learning. Die Wege zum Futter entsprächen dann einer Auswahl von je einem Eintrag pro Spalte, also einer Ameise, die durch die Spalten läuft und dabei Pheromone ablegt. Die Ablage der Pheromone kann allerdings erst nach komplettem Durchschreiten des Wegs geschehen, da die Güte der Lösung (also die Kürze des Weges im übertragenden Sinne) nur für die komplette Lösung durch Simulation festgestellt werden kann. Andere Abbildungstechniken sind denkbar.

Tabu Search. Tabu Search wurde von Fred Glover entwickelt und ist laut ihm selbst ein übergeordnetes heuristisches Verfahren, mit dem es erreicht werden kann, aus lokalen Maxima des Suchraums zu entkommen und ein globales Maximum zu finden [Glo90]. Von einer initialen Lösung, die zufällig erzeugt oder mit einer anderen Heuristik bestimmt wird, werden mit der Nachbarschaftsfunktion, die von der Art der Codierung der Lösung definiert wird, weitere Lösungen generiert. Diejenige der benachbarten Lösungen mit dem besten Fitnesswert wird als neue momentane Lösung gespeichert, um im nächsten Schritt deren Nachbarn zu generieren. Dadurch wird der Suchraum auf einem bestimmten Pfad durchlaufen, bis ein Abbruchkriterium erfüllt ist (z. B. die Anzahl der Iterationen). Lokale Maxima können verlassen werden, aber es wird immer die bisher beste gefundene Lösung gespeichert.

Um Zyklen im Pfad zu vermeiden werden *Tabu-Einschränkungen* benutzt. Wenn z. B. die aktuelle Lösung ein lokales Optimum ist, ist es sinnvoll, beim weiteren Durchlaufen des Suchraums, diese Lösung nicht mehr abzuschreiten. Sie wird als tabu markiert und kann dann in der Folgezeit nicht mehr durchlaufen werden, bis entweder eine bestimmte Zeit abgelaufen ist oder ein sogenanntes *Aspirationskriterium* greift, um das Tabu aufzulösen. Die Aspirationskriterien und die Tabu-Einschränkungen können unterschiedlich definiert werden.

Simulated Annealing. Ein dem Tabu Search ähnliches Optimierungsverfahren ist das Simulated Annealing, vorgestellt in [KGV83]. Wie beim Tabu Search wird ein Pfad durch den Suchraum beschritten, indem immer wieder Nachbarn einer aktuellen Lösung gebildet wer-

den. Auch in diesem Verfahren können lokale Maxima überwunden werden. Hier wird eine benachbarte Lösung als neue aktuelle Lösung übernommen, falls diese einen besseren Fitnesswert aufweist. Mit einer gewissen Wahrscheinlichkeit wird aber auch eine schlechtere Lösung akzeptiert. Diese Wahrscheinlichkeit nimmt im Verlauf der Optimierung immer weiter ab, bis schließlich ein lokales Optimum nicht mehr verlassen werden kann. Das bedeutet, dass anfangs noch große Sprünge von einem lokalen Optimum möglich sind und dann die Dynamik immer weiter abnimmt. Dies stellt eine Nachahmung eines Systems mit beweglichen Molekülen und sich abkühlender Temperatur dar, in dem die Moleküle sich zunächst stark und dann immer schwächer bewegen.

5.4 Untersuchung der Anwendbarkeit von GA auf FSM-kontrollierte Agenten

Anhand eines konkreten Beispiels soll die Effektivität eines GA für die Evolvierung von FSM-kontrollierten Agenten untersucht werden. Die Ergebnisse der Untersuchung wurden teilweise in [EHG11] publiziert. Dabei soll ein prinzipieller GA entworfen werden, der mit unterschiedlichen Mutations- und Rekombinationstechniken sowie anderen Parametern getestet wird. Das Ziel ist es einerseits, herauszufinden, ob der GA grundsätzlich effektiv ist und welche Techniken und Parameter geeignet sind. Bei der Beurteilung der Effektivität werden die gefundenen Lösungen mit der bekannten optimalen Lösung verglichen.

Als Beispielanwendung wird das CEP verwendet. 26 verschiedene Welten (Abb. B.1 und B.2) sollen von den Creatures gelöst werden. Unter Verwendung der Ergebnisse aus [Hal08] konnten alle relevanten 98.869.740 Kandidaten mit $n = 5$ Zuständen auf allen 26 Welten simuliert werden. Der ursprüngliche nicht reduzierte Suchraum hat dabei eine Größe von $|K| = 10^{10}$. Die Simulationen wurden in Software mit einem Java-Programm durchgeführt. Die Berechnungszeit für alle Simulationen auf einem Intel Xeon QuadCore mit 2GHz und 4 parallelen Threads belief sich auf ca. 60 Stunden. Für $n = 6$ ergäbe sich aufgrund der über 14.760.000.000 relevanten Kandidaten bei gleicher Ausrüstung eine Rechenzeit von über einem Jahr.

Das Ziel ist es, möglichst robuste Agenten zu finden, die auf allen 26 Welten möglichst viele Zellen besuchen. Eine geeignete *Fitnessfunktion* bildet jede Lösung auf einen Wert ab, der dieses Ziel repräsentiert. Sie muss dabei einen Vergleich zwischen zwei beliebigen Kandidaten ermöglichen. Der Fitnesswert ist in diesem Fall die Summe aller besuchten Zellen aus allen Welten.

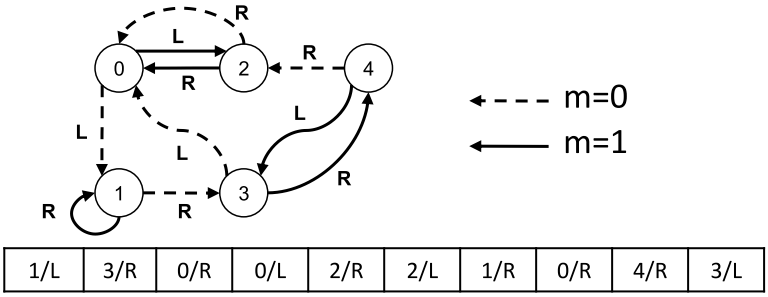

Abb. 5.4.: Die Optimale FSM mit fünf Zuständen.

Insgesamt sind in allen Welten 6227 zu besuchende Zellen enthalten, das ist also der theoretisch maximal möglichen Fitnesswert. Die Resultate der Simulation zeigen, dass kein einziger

der Kandidaten alle 26 Welten lösen konnte. Die optimale Lösung mit 5 Zuständen (Abb. 5.4) hat einen Fitnesswert von $f = 6140$, die zweitbeste $f = 6100$ und die drittbeste $f = 6084$. 21 FSMs haben einen Wert größer als 6000 (Tab. C.1). Die Verteilung der Fitnesswerte über alle relevanten FSMs zeigt, dass die meisten Kandidaten sehr wenige Zellen besuchen können (Abb. 5.5).

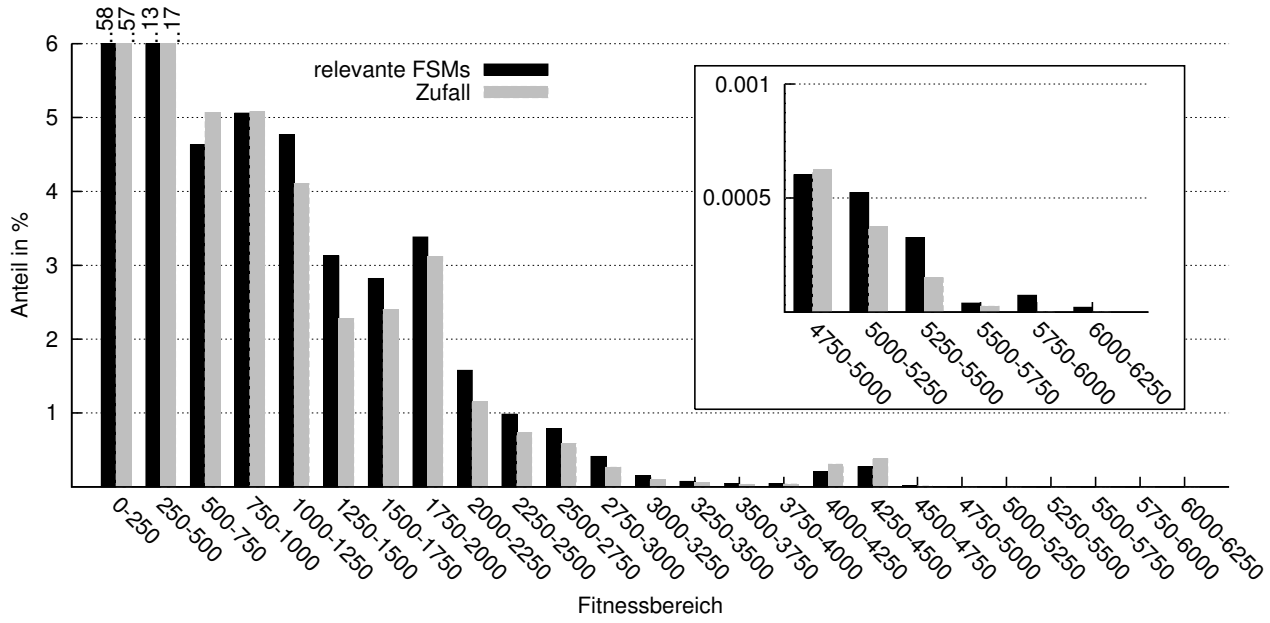

Abb. 5.5.: Fitnessverteilung der relevanten FSMs und zufällig erzeugter FSMs.

Ein Experiment mit zufälliger Suche nach FSMs im gesamten nicht reduzierten Suchraum ergab in etwa die gleiche Verteilung bei 40.000.000 zufällig erzeugten und simulierten FSMs (Abb. 5.5). Das spricht dafür, dass zu jeder FSM aus der reduzierten Menge in etwa gleich viele (äquivalente) FSMs im gesamten Suchraum existieren, die den gleichen Fitnesswert besitzen. Allein durch Zustandspermutationen entstehen zu jeder relevanten FSM 23 weitere mit gleichem Fitnesswert im großen Suchraum. Bei der zufälligen Suche wurde allerdings nur eine einzige FSM mit einem Fitnesswert über 6000 gefunden (mit $f = 6072$), und nur 8 weitere FSMs mit einem Fitnesswert zwischen 5750 und 6000. Die Simulation aller zufällig erzeugten FSMs dauerte ungefähr 24 Stunden.

Für die Konstruktion eines GA müssen die Selektion, die Rekombination, die Mutation und die Substitution und deren Parameter definiert werden. Es gibt eine Vielzahl von variablen Parametern und Techniken für die einzelnen Schritte, die nicht in allen Kombinationsmöglichkeiten getestet werden können. Dennoch sollen hier einige Varianten verwendet und verglichen werden, um einen Anhaltspunkt für eine geeignete Technik zu bekommen. Abb. 5.6 zeigt das Schema des Ablaufs aller Schritte einer einzelnen Iteration des hier verwendeten GA, der im Folgenden beschrieben wird.

5.4.1 Beschreibung des verwendeten Genetischen Algorithmus

Population. Die Populationsgröße ist einer der Parameter, der mit unterschiedlichen Werten in diesem Experiment getestet wurde. Er kann die Werte $|P| \in \{20, 40, 50, 60, 100, 200, 500\}$

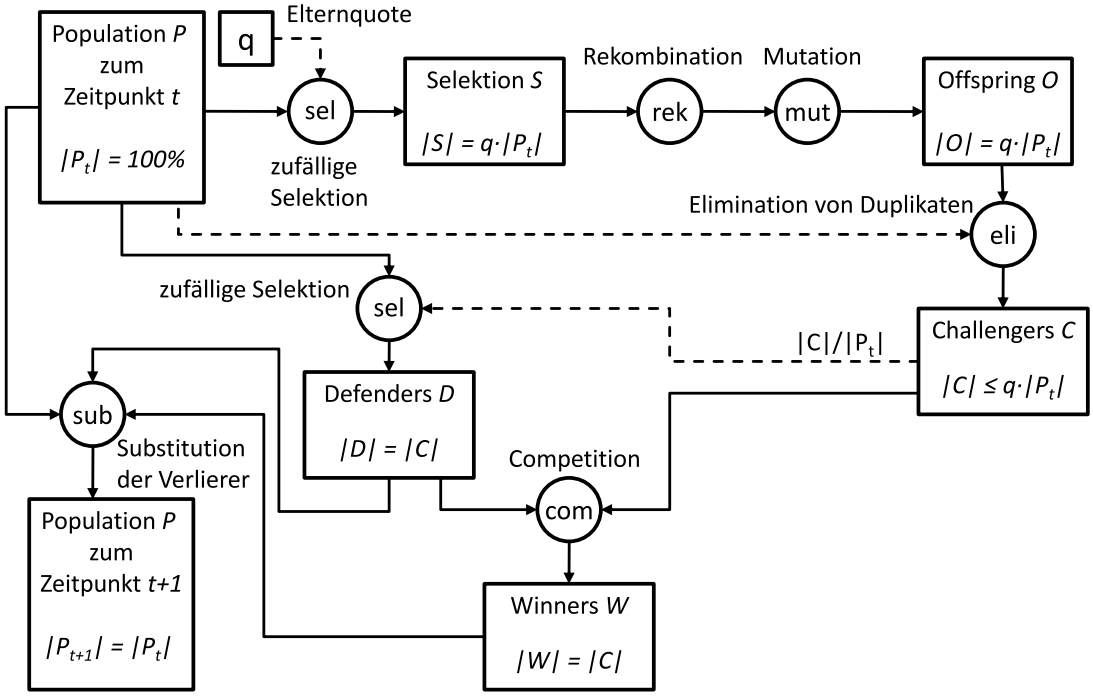

Abb. 5.6.: Schema des GA mit konstanter Populationsgröße.

annehmen. Der GA arbeitet nur auf einer einzigen Liste mit dieser während des Optimierungsvorgangs konstanten Größe.

Selektion. Aus der Population zum Zeitpunkt t wird eine Untermenge S selektiert. Die Größe der Untermenge ist abhängig von der Populationsgröße und einer Elternquote q. In diesem Experiment ist die Elternquote auf $q = 0,2$ festgesetzt worden, was bedeutet, dass 20% der Individuen ausgewählt werden. Die Auswahl der Individuen aus der Population ist komplett zufällig. Die mehrfache Selektion eines Individuums ist aber ausgeschlossen.

Eine Reihe von alternativen Selektionsmechanismen wird in der Literatur beschrieben und analysiert, unter anderem in [BT95]. Populär sind z. B. die *Rouletteradselektion*, die *Turnierselektion*, die *Truncation-Selektion* und die *rangbasierte Selektion*. Bei der Rouletteradselektion wird die Wahrscheinlichkeit einer Auswahl proportional zur Fitness gewählt, so dass ein besserer Kandidat eine höhere Wahrscheinlichkeit hat, als Elter zu fungieren. In [KEFH09] wurde dieser Selektionsmechanismus mit GP für das CEP angewendet. Die Turnierselektion funktioniert so, dass zunächst einige Individuen zufällig ausgewählt werden und aus dieser Zwischenauswahl dasjenige Individuum mit dem besten Fitnesswert in die eigentliche Selektion übernommen wird. Das wird so oft wiederholt, bis die Selektion vollständig ist. Nur die besten Individuen aus der momentanen Population werden bei der Truncation-Selektion ausgewählt. Bei der rangbasierten Selektion werden die Individuen nach ihrer Fitness sortiert und die Selektionswahrscheinlichkeit proportional zu ihrer Position in der Rangliste gewählt.

Grundsätzlich sind bei der Selektion noch andere Mechanismen denkbar, die nicht nur auf der Fitness beruhen, sondern z. B. auch auf der Verweildauer eines Individuums in der Population oder der Häufigkeit der bisherigen Auswahl. Kombinationen von Selektionsmechanismen oder wechselnde Mechanismen abhängig von der Zeit oder anderen Parametern des GA sind ebenfalls möglich.

Rekombination. Aus der Selektion S werden zunächst $|S|/2$ disjunkte Paare gebildet. Ein solches Paar bestehe aus den Individuen A und B. Aus jedem Paar werden zwei Nachkommen generiert. In dieser Untersuchung wird dafür eine von zwei verschiedenen Rekombinationstechniken verwendet (Abb. 5.7). Die erste ist das *Uniform Crossover* (UN), die zweite das *One-Point Crossover* (1P).

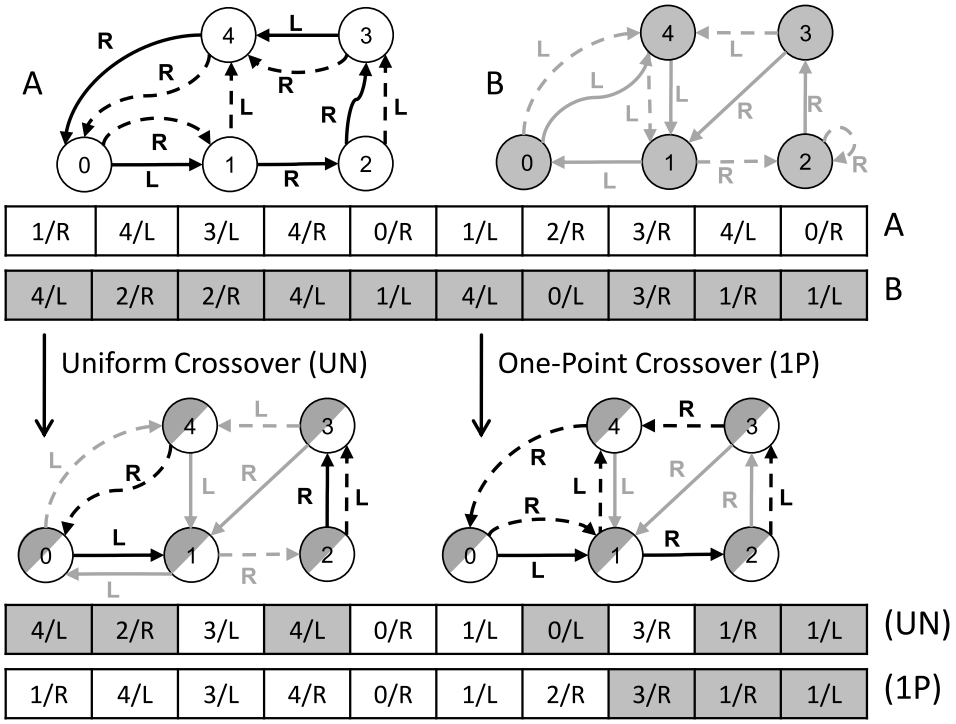

Abb. 5.7.: Beispiele für Rekombinationstechniken. Uniform Crossover links und One-Point Crossover rechts.

Beim UN ist der Vorgang wie folgt. Da die Länge des Genoms aller Individuen gleich ist, können die einzelnen Gene (Paare aus Folgezustand und Output) mit Positionen markiert und gegenübergestellt werden. Für jede Position wird zufällig mit der Wahrscheinlichkeit $p = 0.5$ das Gen von Elter A ausgewählt und mit der Gegenwahrscheinlichkeit das Gen von Elter B. Das jeweils ausgewählte Gen wird an die gleiche Position im Nachkommen geschrieben. Der zweite Nachkomme wird genauso gebildet, nur dass an jeder Position das Gen vom anderen Elter ausgewählt wird. Beim 1P wird stattdessen nur eine Position ausgewählt. An dieser Position werden die Genome von den Eltern getrennt und vertauscht wieder zusammengefügt. Dadurch entstehen zwei neue Nachkommen. In Abb. 5.7 ist jeweils nur ein Nachkomme für jede Technik abgebildet.

Eine Verallgemeinerung der zweiten Rekombinationstechnik ist das *Multi-Point Crossover*. Dabei werden die Eltern an mehreren Positionen getrennt und vertauscht wieder zusammengefügt. Darüber hinaus gibt es auch Rekombinationstechniken, die die einzelnen Gene arithmetisch verknüpfen. Eine große Vielfalt von Techniken oder Varianten der genannten Techniken ist denkbar. So könnte z. B. ein Nachkomme auch aus mehr als zwei Eltern gebildet werden. Es könnte auch ein dominantes Elternteil geben, dessen Gene eine höhere Wahrscheinlichkeit besitzen, ausgewählt zu werden (beim UN). Es ist auch nicht zwingend erforderlich, genau zwei Nachkommen aus jedem Elternpaar zu erzeugen.

Mutation. Jede der durch Rekombination neu erzeugten FSMs durchläuft nun eine Mutation. Dabei wird eine von drei Techniken verwendet. Die erste wird *Single-Gene Mutation* (SG) und die zweite *Every-Gene Mutation* (EV) und die dritte *Double-Gene Mutation* (DB) genannt. Alle drei Techniken basieren darauf, dass ein Gen, also ein Paar aus Folgezustand s' und Entscheidung e, durch ein anderes mit zufällig erzeugten Werten ersetzt werden kann (Abb. 5.8).

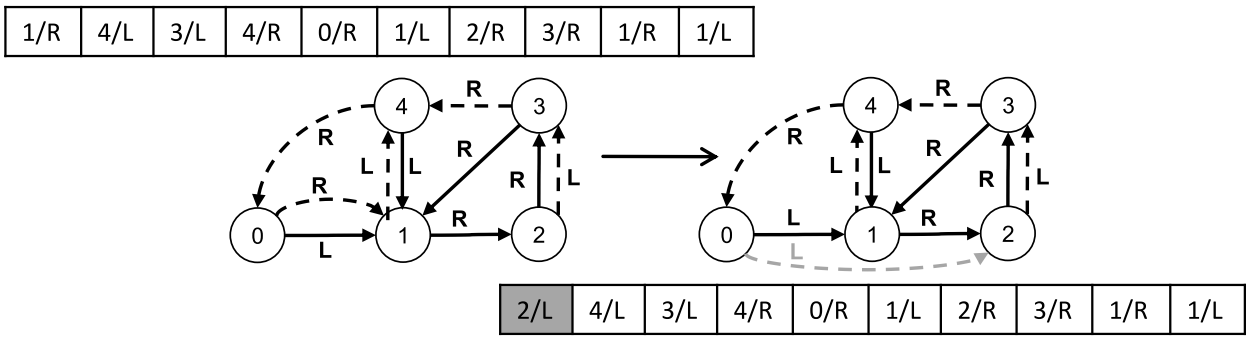

Abb. 5.8.: Beispiel für eine Mutation des Gens an der ersten Position.

Bei der Single-Gene Mutationstechnik wird ein Nachkomme mit der Wahrscheinlichkeit p mutiert und mit der Wahrscheinlichkeit $1-p$ belassen, wie es ist. Für den Fall, dass der Nachkomme mutiert wird, wird eines der Gene zufällig ausgewählt und ersetzt. Bei der Double-Gene Mutationstechnik wird ein Nachkomme ebenfalls mit der Wahrscheinlichkeit p mutiert. Allerdings werden nun zwei Gene zufällig ausgewählt und ersetzt. Bei der Every-Gene Mutationstechnik wird dagegen jedes einzelne Gen mit der Wahrscheinlichkeit p mutiert. In diesem Experiment wird die Technik SG mit den Wahrscheinlichkeiten $p \in \{0,5; 1\}$, die Technik DB mit der Wahrscheinlichkeit $p = 1$ und die Technik EV mit den Wahrscheinlichkeiten $p \in \{0,05; 0,1; 0,2\}$ verwendet. Da jedes Genom aus 10 Genen besteht, wird sowohl bei der Technik SG mit $p = 1$ als auch bei der Technik EV mit $p = 0,1$ im Durchschnitt jedes zehnte Gen mutiert. Die mittlere Anzahl von Mutationen pro Genom (*Mutationshäufigkeit*) beträgt damit 1. In dieser Hinsicht äquivalent sind dann auch die Techniken EV mit $p = 0,05$ und SG mit $p = 0,5$ sowie EV mit $p = 0,2$ und DB mit $p = 1$, mit einer Mutationshäufigkeit von 0,5 bzw. 2. Aus Sicht eines Genoms liegen die Wahrscheinlichkeiten einer Mutation bei den Techniken SG und DB bei 100%. Bei EV sind es $1-(1-p)^{10}$ und damit etwa 40%, 65% oder 90%.

Die Mutationshäufigkeiten in diesem Experiment entsprechen in etwa den Mutationshäufigkeiten aus [JCC+90] (0,5 bis 4,5), wobei die Gene dort einzelne Bits sind und das Genom aus deutlich mehr Genen besteht. Für die Codierung eines Paars würden vier Bits benötigt (10 Möglichkeiten bei 2 möglichen Outputs und 5 möglichen Folgezuständen), so dass bei einer angenommenen Mutation bis zu vier Bits, aber im Mittel zwei Bits, verändert würden. Das ganze Genom könnte mit 40 Bit codiert werden, gegenüber 448 Bits in [JCC+90]. Es würden also im Mittel zwischen 1 und 4 von 40 Bits mutiert, während es in [JCC+90] 0,5 bis 4,5 von 448 Bits sind. Eine ähnliche Größenordnung wurde mit einer Mutationswahrscheinlichkeit von 15% pro Gen (ein Gen ist ein Integer) in [Mar04] verwendet. Das entspreche einer Mutationshäufigkeit von ca. einem Integer pro Genom bei einer variablen Länge des Genoms mit bis zu 64 Integers. In [SG00b] hängt die Mutationswahrscheinlichkeit von der Anzahl der Zustände der FSM ab: $0,001/n$ pro Gen, und erreicht bei angenommenen 5 Zuständen eine Mutationshäufigkeit von 0,05.

Die Mutationswahrscheinlichkeiten dieses Experiments können hoch erscheinen, wenn man Vergleiche mit anderen Anwendungen von GA macht. Deutlich kleinere Mutationswahrscheinlichkeiten wurden ebenfalls im Vorfeld des Experiments getestet, konnten aber keine zufriedenstellenden Resultate erzeugen. Dies könnte unterschiedliche Ursachen haben, z. B. die Wahl anderer Parameter des GA oder die Art der Codierung der FSM oder die Konsistenz des Suchraums. Eine genaue Überprüfung der Ursache ist jedoch nicht Bestandteil dieser Arbeit.

Bei 2 möglichen Outputs und 5 möglichen Folgezuständen gibt es nur 10 zufällig erzeugbare Paare. Mit einer Wahrscheinlichkeit von 10% hat eine Mutation eines Gens also gar keinen Effekt. Mit der Erhöhung der Anzahl der Zustände oder der Outputs verringert sich diese Wahrscheinlichkeit, so dass bei großen FSMs möglicherweise geringere Mutationswahrscheinlichkeiten gewählt werden sollten.

Es existiert eine Vielzahl weiterer Mutationstechniken. Beispielsweise könnte man die zufälligen Änderungen beschränken, indem man entweder nur den Folgezustand oder nur die Entscheidung beliebig ersetzt. Eine ganz andere Art der Mutation ist die Vertauschung der Position zweier Gene, oder die Permutation von Genen in einem Teilstück des Genoms.

Substitution. Nach dem Durchlaufen der Mutation bilden die Nachkommen die Menge O. Diese Menge wird mit den Individuen aus der Population P_t verglichen, um bereits vorhandene Individuen, die nochmal erstellt wurden, löschen zu können. Für die Individuen der eventuell verkleinerten Menge $C = O \setminus P$ wird dann durch Simulation aller Welten der Fitnesswert bestimmt. Abhängig von diesem Fitnesswert wird aus der alten Population und den Nachkommen die neue Population gebildet.

In diesem Experiment wird das mittels einer *Wettbewerbsselektion* gemacht. Die Menge der evaluierten und gegebenenfalls reduzierten Nachkommen ist die Menge der Herausforderer C (engl.: Challengers). Für jeden Herausforderer wird aus der Population zufällig ein Verteidiger ausgewählt. Daraus entsteht die Menge D (engl.: Defenders). Die Fitnesswerte der so entstehenden Paare werden verglichen und das Individuum mit dem jeweils besseren Fitnesswert kommt in die Menge der Gewinner W (engl.: Winners). Nun werden aus der Population alle Verteidiger entfernt und die Gewinner hinzugefügt: $P_{t+1} = (P_t \setminus D) \cup W$. Bei diesem Verfahren ist sichergestellt, dass der beste je gefundene Kandidat niemals aus der Population entfernt werden kann.

Andere Substitutionstechniken sind z. B. das Einsortieren der Herausforderer nach dem Fitnesswert und die anschließende Löschung der schlechtesten Kandidaten, so dass die Population konstant bleibt. Denkbar ist auch eine Population mit veränderlicher Größe oder weitere Substitutionstechniken, wie z. B. eine Wettbewerbsselektion, bei der die Menge der Verteidiger der Menge der selektierten Eltern entspricht $S = D$.

5.4.2 Ergebnisse des Experiments

Für jede der Kombinationen aus den sieben verschiedenen Populationsgrößen, den zwei Rekombinationstechniken und den sechs Mutationstechniken (inklusive Wahrscheinlichkeiten) wurden 20 Durchläufe des GA ausgeführt. In jedem Durchlauf wurde die Anzahl der Iterationen so gewählt, dass 1.000.000 FSMs als Nachkommen erzeugt und getestet wurden, d. h. es wurden $1.000.000/(q \cdot |P|)$ Iterationen ausgeführt. Die Anzahl der getesteten FSMs entspricht $0,01\%$ des gesamten Suchraums, wobei nicht ausgeschlossen ist, dass mehrfach identische Nachkommen erzeugt werden. Die Dauer eines Durchlaufs betrug im Mittel etwa 36 Minuten, was etwa

einem Hundertstel der Zeit entspricht, die für die komplette Aufzählung benötigt wurde. Im Folgenden werden die einzelnen Kombinationen als R-$M_p(|P|)$ notiert, wobei R ein Platzhalter für die Rekombinationstechnik und M_p ein Platzhalter für die Mutationstechnik mit Mutationswahrscheinlichkeit p ist.

Nach der Ausführung des GA wurde für jeden Fall die über die 20 Durchläufe gemittelte Fitness der besten gefundenen FSM sowie die Anzahl der gefundenen Optima ermittelt (Tab. 5.1). In 79 von 84 Fällen wurde in mindestens einem der 20 Durchläufe das Optimum gefunden. Bei den Fällen $1P$-$EV_{0,2}(40)$, $1P$-$EV_{0,2}(50)$ und UN-$EV_{0,1}(40)$ wurde dies sogar in 9 von 20 Durchläufen erreicht. Bei weiterer Betrachtung der Daten zeichnen sich mehrere Tendenzen ab:

- Schlechtere Fitnesswerte und weniger Optima wurden in Fällen mit geringer Populationsgröße und kleiner Mutationswahrscheinlichkeit erreicht.
- Bei geringen Mutationswahrscheinlichkeiten werden mit Uniform Crossover bessere Fitnesswerte erreicht. Bei hohen Mutationswahrscheinlichkeiten fällt die verwendete Rekombinationstechnik nicht mehr so ins Gewicht.
- Obwohl die besten Fitnesswerte mit Populationsgrößen von 40 und 50 erreicht worden sind, ist im Durchschnitt die Populationsgröße 200 am erfolgreichsten. Die Populationsgröße 500 weist insbesondere bei hohen Mutationswahrscheinlichkeiten schlechtere Fitnesswerte und weniger gefundene Optima auf als geringere Populationsgrößen.
- Die Mutationstechnik Every-Gene Mutation erreicht in der Regel bessere Werte als Single-Gene oder Double-Gene, wenn direkt diejenigen Fälle verglichen werden, in denen im Mittel gleich viele Gene mutiert werden.

Tab. 5.1.: Beste Fitness über alle 20 Durchläufe gemittelt und Anzahl der in je 20 Durchläufen des GA gefundenen Optima.

$\|P\|$	$1P$-$SG_{0,5}(\|P\|)$	$1P$-$SG_1(\|P\|)$	$1P$-$DB_1(\|P\|)$	$1P$-$EV_{0,05}(\|P\|)$	$1P$-$EV_{0,1}(\|P\|)$	$1P$-$EV_{0,2}(\|P\|)$
20	5096,2 / 0	5334,8 / 1	5897,9 / 4	5579,5 / 4	6002,2 / 2	6092,2 / 7
40	5578,85 / 1	5472,6 / 0	6002,9 / 4	5732,5 / 2	6051,9 / 6	6106,2 / 9
50	5740,85 / 1	5491,1 / 2	6016 / 3	5819,7 / 1	6011,05 / 4	6092,8 / 9
60	5392,95 / 2	5659,45 / 1	6096,75 / 8	5868 / 3	6017,65 / 2	6082,1 / 7
100	5807,35 / 2	5770,5 / 2	6074,25 / 8	5817,55 / 4	5927,65 / 6	6088,65 / 6
200	5682,8 / 1	5944,75 / 4	6101,9 / 7	5917,25 / 1	6052,25 / 4	6092,95 / 6
500	5970,4 / 1	6034,9 / 3	6035,75 / 2	5782,65 / 0	5938,6 / 2	6066,4 / 2

$\|P\|$	UN-$SG_{0,5}(\|P\|)$	UN-$SG_1(\|P\|)$	UN-$DB_1(\|P\|)$	UN-$EV_{0,05}(\|P\|)$	UN-$EV_{0,1}(\|P\|)$	UN-$EV_{0,2}(\|P\|)$
20	5377,1 / 3	5388,1 / 2	6050,55 / 2	5932,5 / 3	6015,7 / 2	6063,9 / 3
40	5819,65 / 3	5647 / 2	6077,2 / 7	5921,35 / 5	6097,7 / 9	6077,7 / 3
50	5661,55 / 0	5794,7 / 2	6085,6 / 6	5927,35 / 4	6029 / 5	6078,85 / 4
60	5829,95 / 2	5859,95 / 4	6011,9 / 3	5868 / 3	6015,05 / 5	6082,1 / 5
100	6030,2 / 5	5892,6 / 3	6087,75 / 6	5841,55 / 1	6016,95 / 7	6087,1 / 8
200	5939,75 / 3	6075,75 / 7	6089,55 / 6	6081 / 7	6030,7 / 7	6097,75 / 8
500	5988,85 / 1	6016,55 / 2	5996,2 / 1	5950,35 / 2	5974,4 / 2	5942,5 / 0

Um herauszufinden, wie die Fitness sich über die Iterationen des GA entwickelt, wurde die beste bis dahin gefundene Fitness in jeder Iteration protokolliert. Im Fall UN-$EV_{0,2}(200)$, in

dem acht mal das Optimum gefunden wurde, kann man feststellen, dass dies im Schnitt nach etwa 770.000 simulierten FSMs geschehen ist. Eine FSM mit einer Fitness von über 6000 oder knapp unter 6000 wird in den meisten Fällen schon viel früher gefunden (Abb. 5.9). Beim Vergleich mit dem Fall $UN\text{-}SG_{0,5}(200)$ ist letzteres ebenso der Fall, allerdings ist die Streuung am Ende des GA größer. Die Beobachtungen bei den nicht dargestellten Fällen ist ähnlich. Einige weisen eine hohe Streuung auf, aber in mehr als der Hälfte der Durchläufe wird eine Fitness über 6000 oder knapp unter 6000 schon weit vor Ende des GA erreicht. Falls für die Suche nach einer FSM nur wenig Zeit zur Verfügung steht, empfiehlt es sich daher, mehrere kurze Durchläufe parallel zu starten. Bereits nach 160.000 simulierten FSMs wird bei den meisten Techniken in wenigstens einem der 20 Durchläufe eine FSM mit einer Fitness über 6000 gefunden (Tab. 5.2). In der Tendenz scheinen hohe Mutationswahrscheinlichkeiten und eher kleine Populationsgrößen bei kurzen Durchläufen bessere Ergebnisse zu liefern. Je nach Mutationstechnik ist mal UN und mal 1P die bessere Wahl bei der Rekombination. In 16 von 20 Durchläufen werden im Fall $1P\text{-}DB_1(60)$ FSMs mit $f \geq 6000$ gefunden und in 14 von 20 Durchläufen im Fall $1P\text{-}DB_1(20)$. Ganz schlecht mit null Funden in 20 Durchläufen schneiden die Fälle mit Populationsgröße $|P| = 500$, Rekombination UN und Mutation EV, DB und $SG_{0,5}$ ab, sowie auch die Fälle $1P\text{-}EV_{0,05}(50/200)$ und $1P\text{-}SG_{0,5}(20)$.

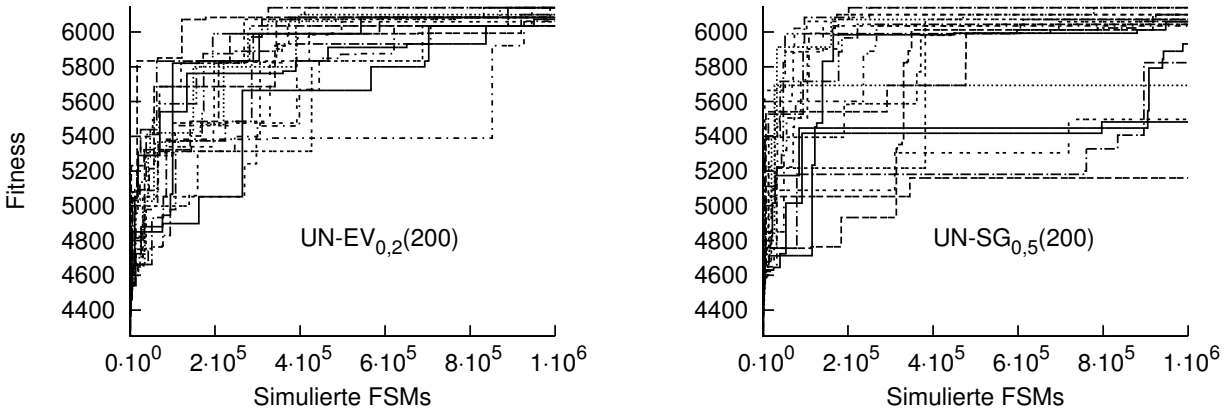

Abb. 5.9.: Verlauf der besten Fitness über die simulierten FSMs im GA am Beispiel der Fälle $UN\text{-}EV_{0,2}(200)$ (links) und $UN\text{-}SG_{0,5}(200)$ (rechts). Alle 20 Durchläufe sind dargestellt.

Tab. 5.2.: Anzahl der Durchläufe, in denen nach 160000 getesteten FSMs, Lösungen mit $f \geq 6000$ gefunden wurden.

	1P-						UN-					
$\|P\|$	$SG_{0,5}$	SG_1	DB_1	$EV_{0,05}$	$EV_{0,1}$	$EV_{0,2}$	$SG_{0,5}$	SG_1	DB_1	$EV_{0,05}$	$EV_{0,1}$	$EV_{0,2}$
20	0	1	14	8	4	11	2	4	8	2	9	9
40	2	2	9	4	7	8	4	5	9	6	7	10
50	4	1	9	0	5	9	5	6	11	3	6	4
60	2	3	16	1	6	6	4	5	3	1	4	7
100	1	4	8	5	6	4	4	6	6	5	4	5
200	3	5	2	0	4	3	3	6	3	2	4	1
500	4	5	4	1	1	1	0	1	0	0	0	0

Wenn man die Streuung der maximalen Fitnesswerte am Ende jedes Durchlaufs betrachtet, kann man Unterschiede zwischen den Mutations- und Rekombinationstechniken ausmachen, die von der Populationsgröße relativ unabhängig sind. In Abb. 5.10 ist die Verteilung der Fitnesswerte für die drei Fälle $1P\text{-}EV_{0,2}(|P|)$, $UN\text{-}EV_{0,2}(|P|)$ und $1P\text{-}SG_{0,5}(|P|)$ dargestellt. Die beiden Fälle mit Every-Gene Mutation weisen eine hohe Anzahl von Fitnesswerten über 6000 auf. Lediglich für die Populationsgröße 500 im Fall mit Uniform Crossover gibt es Durchläufe des GA, die keinen besseren Fitnesswert als 5500 bzw. 5750 finden. Die schlechteste Verteilung von allen Fällen hat $1P\text{-}SG_{0,5}(|P|)$. Insbesondere für kleine Populationsgrößen kann der GA oft keine FSMs mit Fitnesswerten über 5000 finden. Die anderen nicht dargestellten Verteilungen liegen dazwischen.

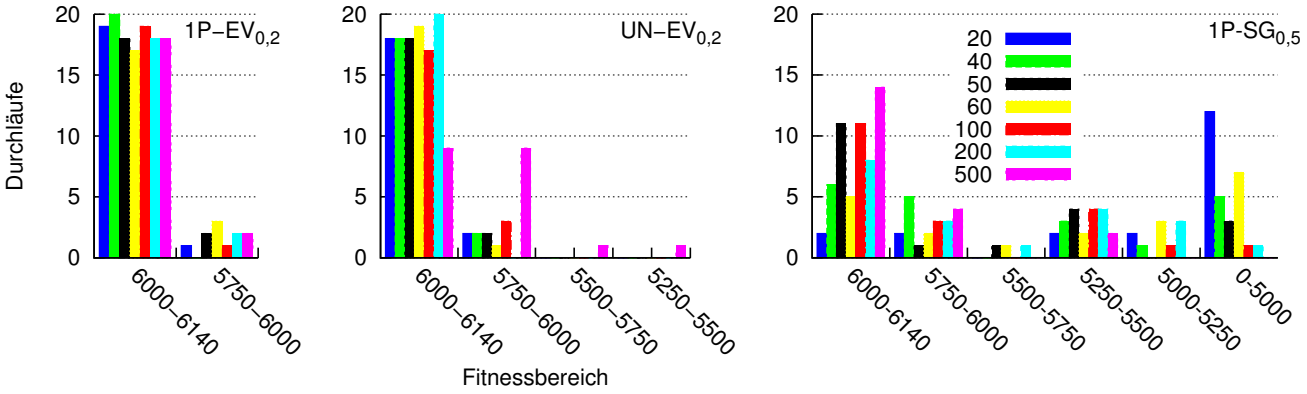

Abb. 5.10.: Verteilung der jeweils besten evolvierten Fitnesswerte aller Durchläufe der Fälle $1P\text{-}EV_{0,2}(|P|)$ (links), $UN\text{-}EV_{0,2}(|P|)$ (mitte) und $1P\text{-}SG_{0,5}(|P|)$ (rechts).

Man findet, die richtige Technik vorausgesetzt, mit sehr hoher Zuverlässigkeit eine FSM mit einer Fitness über 6000, also eine von 21 Möglichen aus 98.000.000 relevanten FSMs, wobei der GA in einem Suchraum von 10^{10} sucht. Grundsätzlich kann man also festhalten, dass man mit bestimmten Rekombinations- und Mutationstechniken sowie Populationsgrößen verlässlich das Optimum oder eine Lösung nahe am Optimum für das CEP finden kann. Kleine Populationsgrößen und zu geringe Mutationswahrscheinlichkeiten eignen sich weniger, ebenso wie zu große Populationen. Die Verwendung des GA stellt gegenüber der vollständigen Aufzählung und der zufälligen Suche eine enorme Zeitersparnis dar, wenn man mit einer Lösung nahe am Optimum zufrieden ist.

5.4.3 Inselmodell des Genetischen Algorithmus

In [ES03, S. 158] wird unter anderem vorgeschlagen, dass man für multimodale Probleme, also solche, die mehrere lokale Optima besitzen, mehrere EAs parallel laufen lässt oder auch gleichzeitig („in tandem“) mit einem Informationsaustausch untereinander. Diese Form von parallel laufenden und interagierenden EAs ist auch als *Inselmodell* (engl.: Island Model EA) bekannt. Die Ergebnisse aus Abschn. 5.4 und die Tatsache, dass in dem relativ inkonsistenten Suchraum viele äquivalente, aber nicht unbedingt benachbarte FSMs existieren (also viele lokale Optima), sprechen dafür, dass ein solches Inselmodell gute Ergebnisse liefern könnte.

In [EHH09] wurde ein Inselmodell verwendet, um dessen Effektivität im Vergleich zu einer vollständigen Suche zu untersuchen. Als Beispielanwendung wird wieder das CEP verwendet.

Fünf Welten aus [Hal08] mit jeweils acht Agenten sollen gelöst werden (Abb. 5.11). Wie in Abschn. 5.4 wurden alle relevanten Kandidaten mit $n = 5$ in Java simuliert. Die Berechnungszeit dafür belief sich auf etwa 165 Stunden (Intel Core2Duo mit 2,4GHz und 2 parallelen Threads).

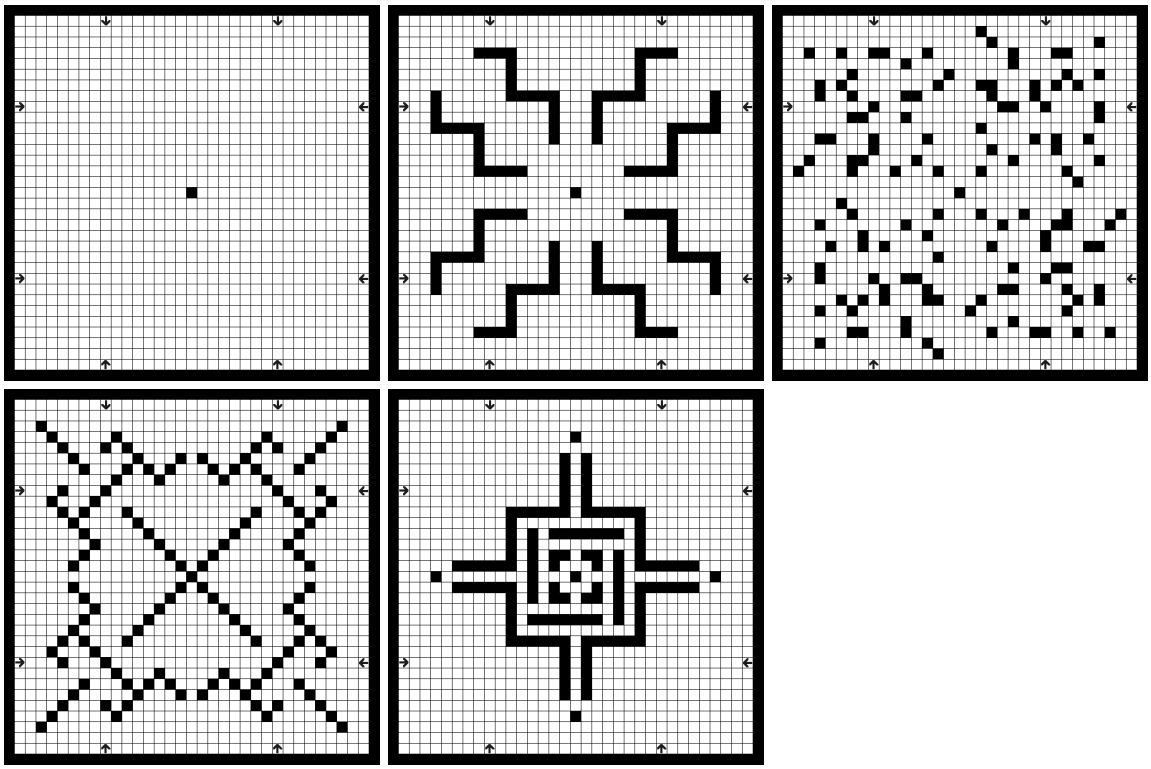

Abb. 5.11.: Initiale Konfigurationen der fünf Welten des CEP mit je acht Agenten. Die Welten sind punktsymmetrisch aufgebaut, also hat jeder Agent während der gesamten Simulation die gleichen Inputs (wie in [Hal08]).

Anders als bei den 26 Welten aus Abschn. 5.4 konnten FSMs gefunden werden, die alle fünf Welten komplett lösen konnten. Dementsprechend würde eine Fitnessfunktion, die nur die Anzahl der besuchten Zellen berücksichtigt, keine Differenzierung zwischen diesen FSMs zulassen. Neben der Robustheit sind aber auch FSMs gesucht, die die Aufgabe möglichst schnell erledigen. Deshalb soll die Anzahl der Generationen g, die zur Lösung gebraucht werden, ebenfalls in der Fitness enthalten sein. Die Fitness einer FSM ist die Summe von einzeln berechneten Werten für jede Welt j: $\sum_{j=1}^{5} f_j$. Die einzelnen Summanden berechnen sich nach folgender Formel:

$$f_j = \begin{cases} pen_{suc} + \bar{v}_j + pen_{lim} & \text{, falls } suc_j = 0 \wedge \bar{v}_j > lim \\ pen_{suc} + \bar{v}_j & \text{, falls } suc_j = 0 \wedge \bar{v}_j \leq lim \\ \check{g}_j & \text{, falls } suc_j = 1 \end{cases}$$

wobei $\check{g}_j$ die Anzahl der benötigten Generationen angibt, falls die Welt erfolgreich gelöst wurde ($suc_j = 1$). Wurde die Welt nicht erfolgreich gelöst ($suc_j = 0$), wird eine Strafe pen_{suc} (engl.: penalty) addiert. $\bar{v}$ gibt die Anzahl der Zellen an, die in der Welt nicht besucht worden sind. Überschreitet diese Zahl eine Grenze lim (engl.: limit), wird eine weitere Strafe pen_{lim} addiert. Ein kleinerer Wert bedeutet hier eine bessere Fitness. Diese Formel repräsentiert eine Dominanzrelation, die eine FSM mit mehr erfolgreich gelösten Welten immer besser bewertet. Das ist deshalb so, weil die Simulation nach 10.000 Generationen ohne Erfolg (oder nach

2.000 Generationen ohne neue besuchte Zelle) abgebrochen wird und der Wert für die Strafe auf $pen_{suc} = 20.000$ gesetzt wurde. Die anderen Werte sind konstant bei $lim = 500$ und $pen_{lim} = 450$. Bei FSMs, die alle Welten erfolgreich lösen besteht der Fitnesswert also nur noch aus der Anzahl der dafür benötigten Generationen.

Insgesamt gibt es 4928 freie Zellen in den fünf Welten. Würde jeder Agent in jeder Generation eine neue Zelle besuchen, bräuchten sie in der Summe 616 Generationen für die Lösung. Dies ist also der theoretisch beste Fitnesswert. Die optimale Lösung mit fünf Zuständen hat einen Fitnesswert von $f = 3284$, die zweitbeste $f = 3566$ und die drittbeste $f = 3857$. 11 FSMs können alle fünf Welten lösen (Tab. C.2). In einer weiteren Aufzählung aller relevanten FSMs wurde ermittelt, dass gar keine FSM mit vier Zuständen alle fünf Welten lösen kann.

Beschreibung des Inselmodells. In diesem Experiment wurde eine Populationsgröße von $|P| = 100$ pro Insel verwendet. Die Anzahl der Populationen ist auf $\#P = I = 7$ parallel laufende Inseln festgelegt worden. In jeder Population werden mit der Elternquote $q = 0,2$ zufällig 20 Lösungen ausgewählt. Jeder der 20 Eltern wird mit der Wahrscheinlichkeit p aus der eigenen Population ausgewählt und mit der Wahrscheinlichkeit $1 - p$ aus einer beliebigen anderen Insel. Durch diese *Migration* kann bei in lokalen Optima festhängenden Populationen ein neuer Impuls geschaffen werden, um aus diesem lokalen Optimum herauszukommen. Zu große *Migrationswahrscheinlichkeiten* sorgen allerdings eher dafür, dass alle Populationen die gleichen lokalen Optima finden. Bei zu kleinen Werten ist der Effekt möglicherweise zu gering. Die Migrationswahrscheinlichkeit wird hier auf $p = 0,02$ gesetzt.

Aus den 20 selektierten Eltern werden 10 Paare gebildet. Jedes Paar erzeugt genau einen Nachkommen per UN und anschließender Mutation per $SG_{0,09}$. Das ist im Vergleich zum Experiment in Abschn. 5.4 eine sehr geringe Mutationswahrscheinlichkeit. Aber durch die neu hinzugekommene Migration sollen ja ebenfalls lokale Optima überwunden werden, so dass hier eine Kompensation stattfinden könnte. Nach der Elimination von Duplikaten wird per Simulation der Fitnesswert der bis zu 10 Nachkommen bestimmt. Aus den Nachkommen und den 100 Kandidaten aus der momentanen Population wird die neue Population mit 100 Kandidaten gebildet, indem die Lösungen mit den schlechtesten Fitnesswerten aussortiert werden.

Die Migration zwischen den Inseln kann auch anders modelliert werden [ES03, S. 158f]. Beispielsweise kann man zu bestimmten Zeitpunkten, nicht unbedingt in jeder Iteration, eine Migration vornehmen. Oder man macht einen direkten Austausch von Kandidaten aus zwei Inseln. Man kann die Inseln auch in einer bestimmten (räumlichen) Struktur anordnen, so dass ein Austausch nur zwischen direkten Nachbarn stattfinden kann usw.

Ergebnisse des Experiments. Es wurden zehn voneinander unabhängige Durchläufe des Inselmodell-GA durchgeführt [4]. In jedem Durchlauf wurde eine Anzahl von 40.800 Iterationen gewählt, so dass bei 70 Nachkommen (Duplikate nicht mitgerechnet) pro Iteration in jedem Durchlauf 2.856.000 FSMs getestet wurden. Das entspricht 0.029% des gesamten Suchraums. Ein Durchlauf dauerte ungefähr 5 Stunden, etwas weniger als ein Dreißigstel der Zeit, die für die komplette Aufzählung benötigt wurde.

Die beste FSM, die gefunden werden konnte, war die optimale Lösung in nur einem einzigen Durchlauf nach 22.498 Iterationen. Aber immerhin in fünf von zehn Durchläufen konnte die insgesamt zweitbeste FSM mit $f = 3566$ gefunden werden. Der Zeitpunkt, an dem das geschah, variierte allerdings stark zwischen Iteration 16.100 und 40.034 (Abb. 5.12). In den meisten Durchläufen wird relativ schnell eine erfolgreiche Lösung gefunden, in sechs Fällen vor der

[4] In [EHH09] sind nur sechs Durchläufe angegeben, da zum Zeitpunkt der Veröffentlichung nicht alle Durchläufe fertig waren.

3500. Iteration, in den anderen vier Fällen zwischen Iteration 11.194 und 22.184. Erfolgreich bedeutet hier erfolgreich auf allen fünf Welten.

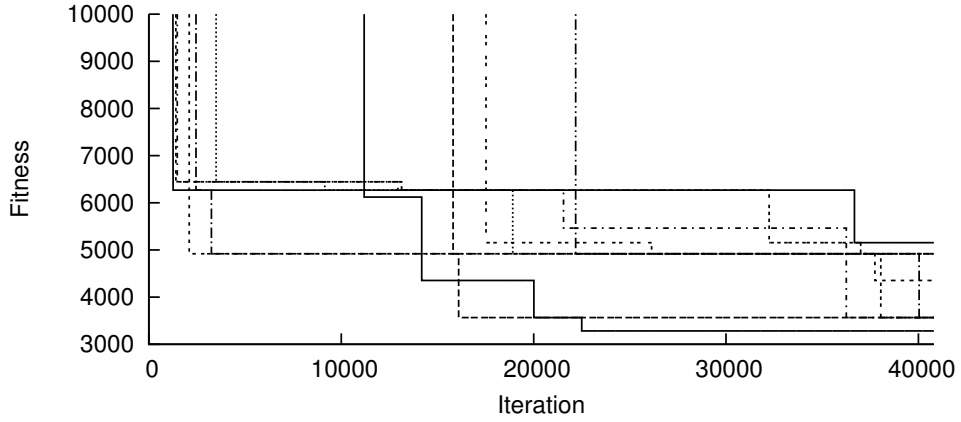

Abb. 5.12.: Verlauf der besten Fitness aller zehn Durchläufe über die Iterationen im Inselmodell-GA mit 5 Zuständen.

Das gleiche Experiment wurde für $n = 6$ und $n = 7$ nochmal durchgeführt, um herauszufinden, ob der GA auch mit größeren Genomen erfolgreiche Lösungen findet, und ob es Effekte bei der Laufzeit gibt. Ein Vergleich mit dem globalen Optimum ist nicht möglich, da es nicht bekannt ist. Die wesentlichen Ergebnisse sind in Tab. 5.3 zusammengefasst. Dass es bessere bzw. mindestens gleich gute Lösungen mit zunehmender Anzahl an Zuständen gibt, ist ein trivialer Schluss, da alle Lösungen mit weniger Zuständen eingeschlossen sind. Aber ob der GA diese besseren oder gleich guten Lösungen auch findet ist nicht klar, da sich der Suchraum exponentiell vergrößert. In diesem Experiment war es aber der Fall. Außerdem war die Laufzeit für alle 40.800 Iterationen sogar geringfügig kleiner. Die Zeitpunkte der jeweils besten Funde (mit $f = 3284$ und $f = 2932$) liegen tendenziell früher, wenn FSMs mit sieben Zuständen evolviert worden sind, allerdings wird der Wert auch nur halb so oft im Intervall bis 40.800 Iterationen gefunden.

Tab. 5.3.: Die besten FSMs und die Anzahl der erfolgreichen FSMs des Inselmodell-GA für 5, 6 und 7 Zustände.

n	Lauf-Zeit	beste Fitness	Anzahl Funde	Fundintervall (Iterationen)	Erfolgreiche FSMs in Durchlauf 1	2	3	4	5	6	7	8	9	10
5	5 Std.	3284	1/10	22498	3	5	7	4	6	5	2	4	5	11
6	4,8 Std.	3284	9/10	9234 - 38594	67	42	53	83	42	55	52	36	60	67
7	4,7 Std.	2932	4/10	10314 - 21098	>100									

Insgesamt sind in jedem Durchlauf erfolgreiche FSMs gefunden worden. Im Fall $n = 5$ zwischen 2 und 11 von 11 möglichen FSMs. Die Anzahl der erfolgreichen FSMs mit 6 und 7 Zuständen ist nicht bekannt. Gefunden werden allerdings deutlich mehr als bei der Optimierung mit 5 Zuständen. Die Ergebnisse weisen darauf hin, dass es prinzipiell möglich ist, auch größere FSMs mit GA zu evolvieren. Die Anzahl der Zustände zu erhöhen konnte in diesem Experiment zu einer insgesamt besseren Lösung und verlässlicher zu erfolgreichen Lösungen führen, ohne

dabei die Suche zu verlängern. Ob es eine Grenze gibt, ab welcher eine weitere Erhöhung der Anzahl der Zustände zu einer Verschlechterung der Effektivität des GA führt, ist nicht klar.

5.4.4 Inkrementeller Genetischer Algorithmus

Aufgrund der Anzahl und Größe der Welten sowie der Anzahl der Agenten darin kann eine Simulation des MAS sehr rechenintensiv werden. Um diesen Aufwand zu reduzieren, kann ein *Inkrementeller GA* (IGA) eingesetzt werden. Das bedeutet, dass der GA zunächst für ein einfaches Problem eingesetzt wird. Dieses einfache Problem besteht aus einer Teilmenge eines komplexeren Gesamtproblems, z. B. nur ein Teil der Welten oder kleinere, schneller simulierbare Welten. Die optimierten Lösungen werden danach als initiale Population in einem GA für ein schwierigeres Problem verwendet werden, bis man schließlich im letzten Schritt eine Optimierung für das komplexe Gesamtproblem vornimmt. Dadurch erhofft man sich einen Effizienzgewinn.

Die Idee, einen GA inkrementell einzusetzen ist nicht neu und wird unter anderem in [MAE06] untersucht. In [SAML08] wird ein IGA zur Optimierung eines Fuzzylogik-gesteuerten Roboters verwendet, der Hindernissen ausweichen soll. Der Controller des Roboters wird dabei nicht direkt für die ganze Umgebung mit Hindernissen evolviert. Stattdessen wird die Umgebung in kleinere einfache Teile unterteilt und schrittweise bis zu einer komplexen Umgebung evolviert. In [EHD10] wird ein IGA zur Optimierung von Agenten im ARP eingesetzt.

Eine weitere Überlegung führt zu noch einer weiteren Form des IGA. Anstatt zunächst simple (oder wenige) und dann komplexe (oder viele) Welten zu verwenden, wird nicht das Problem angepasst, sondern die Komplexität der Lösung. Das bedeutet, dass zunächst FSMs mit wenigen Zuständen evolviert oder sogar aufgezählt werden könnten, und diese Lösungen, dann mit einem Verfahren um einen oder mehrere Zustände erweitert werden könnten, um sie als initiale Population für die nächste Stufe des GA zu verwenden.

Die Ergebnisse aus Abschn. 5.4.3 sprechen allerdings nicht dafür, dass das Evolvieren von FSMs mit weniger Zuständen weniger Zeit benötigt. Ein zweites Fragezeichen steht hinter dem Verfahren, das mehr Zustände zu den FSMs hinzufügt. Wie soll man geschickt aus einer n-Zustands-FSM eine $(n+1)$-Zustands-FSM erstellen? Gibt es vielleicht sogar ein Verfahren, mit dem man aus einer oder mehreren n-Zustands-FSMs, eine FSM mit $2 \cdot n$ oder n^2 Zuständen erzeugt, welche die guten Eigenschaften der kleineren FSMs beibehält? Grundsätzlich lautet die Frage: Kann man kleinere FSMs als Grundlage für eine komplexere Kontrollfunktion (evtl. bestehend aus mehreren oder größeren FSMs) verwenden? In Abschn. 5.5 werden einige Techniken beschrieben, die auch aus diesen Fragestellungen abgeleitet sind.

5.5 Optimierung durch Anpassung der Struktur der Kontrolleinheit

Die Kontrolleinheit des Agenten wurde in Abschn. 3.1.2 als Gesamtheit bestehend aus FSM, Sensor und Aktuator definiert. Alle drei genannten Bestandteile der Kontrolleinheit können strukturell und inhaltlich modifiziert werden, mit dem Ziel, das Agentenverhalten zu optimieren. Es können auch weitere Elemente hinzugefügt werden. Die Struktur der Kontrolleinheit legt fest, welche Aktionen der Agent ausführen kann, welche Eingaben er erhält und welcher Mechanismus anhand dieser Eingaben eine Aktion auswählt. Grundlage aller in diesem Abschnitt vorgestellten Optimierungstechniken ist immer die Verwendung von einer oder meh-

reren FSMs in der Kontrolleinheit. Der Inhalt der Kontrolleinheit spezifiziert das Verhalten des Agenten exakt, indem konkrete Werte in den Mechanismus implementiert werden, z. B. in die Zustandsübergangstabelle der FSM. Der Inhalt kann entweder gar nicht, teilweise oder komplett manuell angegeben werden. Die Teile, die nicht manuell spezifiziert worden sind, können dann mit geeigneten heuristischen Verfahren optimiert werden (Abschn. 5.3).

Da die Kontrolleinheit Schnittstellen zur Welt der Agenten besitzt (über den Sensor und den Aktuator bzw. die Basisoperationen), sind dies betreffende Optimierungen auch abhängig von der Problemstellung. Z. B. kann es vorkommen, dass die Problemstellung nicht erlaubt, dass der Agent durch eine größere (oder kleinere) Aktionsmenge optimiert wird. Die Aktionsmenge ist dann für den Entwickler unveränderlich. Es gibt also je nach Problemstellung Teile der Struktur und des Inhalts der Kontrolleinheit, die nicht optimiert werden können. Die Anwendbarkeit der Techniken zur Optimierung in Abhängigkeit der Problemstellung wird in jedem der folgenden Abschnitte diskutiert.

5.5.1 Variation der Kapazitäten der Kontrolleinheit

Variationen der Anzahl der Zustände der FSMs. Wenn eine FSM zur Verhaltenssteuerung eingesetzt wird, ist zunächst noch offen, wie genau die Struktur der Zustandsübergangstabelle aussehen soll, die Inhalte sollen dann später heuristisch optimiert werden. Angenommen, die Aktionsmenge ist festgelegt und auch die Inputs der FSM sind nicht veränderlich, dann fehlt noch eine Größe, um die Struktur der Tabelle eindeutig zu bestimmen: die Anzahl der Zustände der FSM.

Je mehr Zustände eine FSM haben kann, desto größer wird der Suchraum und desto schwieriger wird die Suche nach dem Optimum. Der Suchraum wächst mit der Anzahl der Zustände n sogar exponentiell. Deshalb kann es sinnvoll sein, n auf ein Maximum zu beschränken, wie z. B. in Abschn. 5.4.3.

Ein Argument für die Erhöhung der Anzahl der Zustände ist, dass dadurch die Intelligenz, sprich die Merkfähigkeit, der Agenten erhöht wird. Zumal alle FSMs mit weniger Zuständen in der Menge der FSMs mit mehr Zuständen enthalten sind, werden dadurch unter Umständen neue und bessere Lösungsstrategien möglich, ohne dass bereits mit wenigen Zuständen mögliche Strategien wegfallen.

An den Grundvarianten, die in Kap. 4 beschrieben worden sind, ändert diese Variation nichts. Sie ist für alle Beispielsysteme einsetzbar. In [KF09, KF10] wurden Agenten mit 1 bis 8 Zuständen für das CEP evolviert und dabei gleichzeitig die Anzahl der Agenten variiert. Nicht immer führte eine Erhöhung der Anzahl der Zustände zu einer Verbesserung der Lösung.

Anpassung des Sensors. Die Möglichkeiten der Agenten, ein Problem zu lösen, sind nicht nur durch die Kontrollfunktion beschränkt. Schon die Daten, die der Kontrollfunktion zur Verfügung stehen, um eine Situation zu analysieren und richtig einschätzen zu können, stellen eine Beschränkung dar. Je weniger Informationen ein Agent von der Umwelt bekommt, desto weniger adäquat kann er einschätzen, welche Aktionen am ehesten zur Problemlösung führen. Andererseits können zu viele Informationen dazu führen, dass die Kontrollfunktion mit der Verarbeitung überfordert ist. Oder anders ausgedrückt, die Kontrollfunktion wird komplexer, wenn sie mehr Inputs verarbeiten muss und dadurch wird eine heuristische Suche nach einem geeigneten Agentenprogramm schwieriger. Daher ist es nicht in jedem Fall trivial, den perfekten Sensor zu entwickeln, sofern dieser nicht durch die Problemstellung fix vorgegeben ist. Ist er

fest vorgegeben, kann man die Inputs nicht mehr erweitern, aber eine Reduzierung der Inputs für die FSM ist noch möglich.

Die Inputreduktion über eine Tabelle wurde in Abschn. 3.1.2 beschrieben. Eine Anpassung dieser Tabelle (IRT) durch Veränderung der Inhalte ist ohne Eingreifen in die Struktur der Kontrolleinheit möglich. Wenn die Anzahl der Inputs für die FSM erhöht werden soll, muss allerdings die Schnittstelle zwischen IRT und FSM verändert werden, damit mehr Inputs verarbeitet werden können.

Nach den Definitionen der Beispielsysteme in Kap. 4 stellt die Veränderung des Sensors oder im Speziellen der IRT eine Veränderung der Grundvarianten dar. Diese Form der Optimierung ist also nur möglich, wenn die Art der Problemstellung dies zulässt.

Lokales Feedback über Veränderungen der Attribute. Eine andere Art von zusätzlichem Input kann auch auf lokaler Ebene, d. h. in diesem Fall innerhalb des Agenten generiert werden. Der Agent kann über den Sensor seine Attribute wahrnehmen, aber er kann nicht die Veränderung seiner Attribute wahrnehmen, da der Sensor kein Gedächtnis besitzt. Lediglich über den eigenen Zustand kann eine Art implizites Gedächtnis geschaffen werden. Ein lokales Feedback, das die Veränderung eines Attributs anzeigt, und als Input für die FSM verwendet wird, stellt eine Alternative dazu dar, die ein Gedächtnis mit expliziter Bedeutung im Agenten einrichtet. In [EH10a] wurde für den ATAC ein solches Feedback eingesetzt, dass die Veränderung des Kommunikationsvektors angezeigt hat.

Erweiterung der Aktionsmenge der Agenten. In den in dieser Arbeit definierten Beispielsystemen sind die Aktionsmengen der Agenten vorgegeben. Von diesen Grundvarianten abweichende Systeme können Agenten beinhalten, die mehr Aktionen ausführen können. Beispielsweise kann der Agent in der Grundvariante des CEP nur Drehungen nach rechts und links ausführen. In bestimmten Situation könnte es aber von Vorteil sein, dass der Agent sich auch um 180 Grad dreht oder seine Richtung beibehält. Eine Erweiterung der Aktionsmenge kann also zu einer Verbesserung des Verhaltens führen.

Der Nachteil der Erweiterung ist, dass einerseits mehr Ressourcen in der Zellstruktur benötigt werden und dass andererseits der Suchraum größer und damit die heuristische Optimierung schwieriger wird. Unter Berücksichtigung dieser Kosten ist es oft nicht einfach zu entscheiden, welche Aktionsmenge optimal ist.

Prinzipiell kann in jedem der drei Beispielsysteme die Aktionsmenge verändert werden. Aber auch hier muss die Problemstellung entsprechend gestaltet sein. Es macht keinen Sinn, die Aktionsmenge zu erweitern, wenn damit Agenten optimiert werden sollen, die physikalischen Beschränkungen unterliegen, die die neuen Aktionen nicht zulassen, sei es durch die eigene Struktur oder die Beschaffenheit der Umwelt. In diesem Fall kann es aber trotzdem möglich sein, die Aktionsmenge von der physikalisch realisierbaren Menge auf weniger Aktionen zu reduzieren, um schneller optimieren zu können oder Ressourcen zu sparen. Generell ist zu beachten, dass in einigen Fällen nicht nur der Aktuator, sondern auch die Umwelt und der Sensor angepasst werden müssen. In [EH09b] wird z. B. unter anderem ein Vergleich zwischen gerichteten und ungerichteten Agenten durchgeführt, die zwangsläufig ganz unterschiedliche Aktionsmengen besitzen und verschiedene Wahrnehmungen und Attribute brauchen (Abschn. 3.1.3). Ein weiteres Beispiel wird im Folgenden erklärt.

Erweiterung der Kommunikationsfähigkeiten der Agenten. Ein Spezialfall der Erweiterung der Aktionsmenge ist die Erweiterung der Kommunikationsfähigkeiten der Agenten unter Einbeziehung der Umwelt. Die Agenten sollen das Konzept der Stigmergie (s. auch Abschn. 2.1.4) verwirklichen. Dies soll geschehen, indem die Agenten *Flags* auf den Zellen lesen und verän-

dern. Mindestens eine neue Aktion muss in diesem Fall zur Aktionsmenge hinzugefügt werden: Das Verändern des Flags. Da das Flag binär ist, kann diese Aktion daraus bestehen, den Wert zu wechseln. Ebenso gut könnten die Agenten aber auch das Flag explizit auf einen Wert setzen. Dann sind entsprechend zwei neue Aktionen notwendig. Anstelle eines Flags kann auch ein nichtbinäres Signal (z. B. Pheromon) verwendet werden, was entsprechend noch mehr mögliche Aktionen notwendig macht. Darüber hinaus muss geklärt werden, ob der Agent gleichzeitig ein Flag modifizieren und sich bewegen können soll. In [EH09d] wurden für den ATAC mit Kommunikation über Flags mehrere Aktionsmengen untersucht und verglichen.

Aber nicht nur die Aktionsmenge muss erweitert werden. Das Flag selbst ist ein Teil der Umwelt bzw. jeder Zelle. D. h. es muss in jeder Zelle ein neues Attribut F hinzugefügt werden. Die Zellregel muss angepasst werden, damit die Aktionen tatsächlich die Werte des Attributs ändern. Schließlich muss der Sensor angepasst werden, damit die Flags überhaupt wahrgenommen werden können. Die notwendigen Änderungen der Struktur gegenüber der Grundvariante hängen von der Art der Information (Flag, Pheromon), der Art der Modifikation (Wechsel oder Explizit, mit/ohne gleichzeitige Bewegung) und der Wahrnehmbarkeit ab (z. B. lesen aus Frontzelle oder eigener Zelle).

Für die Kontrollfunktion bedeutet das, dass mehr Inputs verarbeitet werden müssen und mehr Outputs generiert werden können. Dadurch vergrößert sich der Suchraum und die Bestimmung der optimalen FSM wird schwieriger.

Die Bedeutung der Flags bzw. die Art der Information, die die Agenten aus dem Flag gewinnen ist nicht vordefiniert. Sie stellt zunächst nur ein Mehr an unbestimmter Information dar. Erst durch die Optimierung der FSM ergibt sich möglicherweise ein Verhalten, aus dem man durch Beobachtung schließen kann, was die Agenten mit der Information aus den Flags machen bzw. wie sie sie verarbeiten und so nutzen können, dass das Problem besser gelöst wird. Es kann aber auch eine Form von Stigmergie entstehen, die nicht einfach auf eine bestimmte Bedeutung reduzierbar ist.

5.5.2 Separate FSMs für spezielle Teilaktionen

Agenten entscheiden sich für eine Drehrichtung, zwischen Bewegungen auf eine andere Zelle und dem Verharren auf einer Zelle, sie entscheiden sich für das Setzen oder Löschen von Flags/-Pheromonen, sie entscheiden über die Weitergabe von Informationen an andere Agenten usw., aber bisher haben sie nur eine FSM mit einem einzigen Zustand zur Steuerung all ihrer Entscheidungen. Beispielsweise sind eine Drehung und das Verändern von einem Flag durch diese FSM aneinander gekoppelt. Es gibt nur einen Zustand, von dem beides abhängt, und die gleichen Inputs werden verwendet, um eine Drehung und eine Veränderung des Flags zu steuern. Es kann aber sinnvoll sein, die Drehungen und das Verändern des Flags (zumindest intern in der Kontrolleinheit) voneinander zu trennen, da sie semantisch nicht zusammengehören und auch nicht von den gleichen Bedingungen (Inputs) abhängen sollen.

Die getrennten Entscheidungen kann man als *Teilentscheidungen* der gesamten Entscheidung ansehen. Entscheidet sich der Agent beispielsweise für eine Drehung nach links und das Setzen des Flags auf 1, so wird daraus die Gesamtentscheidung „nach links drehen und Flag setzen“. Die aus den Teilentscheidungen vom Aktuator erstellten Aktionen können dann als *Teilaktionen* bezeichnet werden. Diese sind nicht mit den Basisoperationen gleichzusetzen.

Die Idee zur Trennung der Teilentscheidungen ist die Verwendung je einer FSM $(A, B, \ldots)$ für jede Teilentscheidung (Abb. 5.13). Dadurch erhält man spezielle FSMs für spezielle Arten von Entscheidungen (Fähigkeiten), die man semantisch voneinander trennen kann. Es gibt dann je einen Zustand für jede der getrennten Fähigkeiten $(s_A, s_B, \ldots)$ und unterschiedliche Inputs $(x_A, x_B, \ldots)$, die der Sensor aus den eingehenden Daten der Umwelt berechnet. Die Inputs für beide FSMs können, müssen aber nicht verschieden sein. Die Entscheidung e besteht aus dem Tupel aller Teilentscheidungen $(e_A, e_B, \ldots)$, der Aktuator bildet auf die Aktion $a = (a_A, a_B, \ldots)$ ab. Um im Beispiel von eben zu bleiben, gibt es nun eine „Dreh-FSM“ und eine „Flag-FSM“.

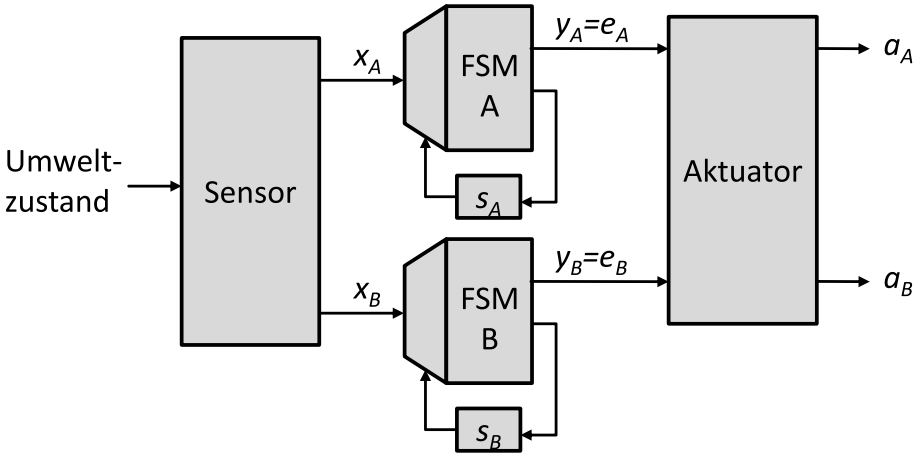

Abb. 5.13.: Struktur der Kontrolleinheit mit multiplen (hier zwei) FSMs für Teilentscheidungen.

Die Verwendung eines GA zur Optimierung ist auch mit mehreren FSMs möglich. Allerdings muss die Codierung des Genoms alle FSMs beinhalten, denn eine Lösung kann nur aus einem Tupel aller FSMs bestehen. Für sich alleine macht eine FSM, die nur Teilentscheidungen trifft, keinen Sinn. Die Lösung zur Anpassung des Genoms ist eine Konkatenierung der Strings aller FSMs. Der Suchraum ist dann abhängig von den Kombinationsmöglichkeiten der Inputs $i_A, i_B, \ldots$, den Kombinationsmöglichkeiten der Outputs $o_A, o_B, \ldots$ und der Anzahl der Zustände $n_A, n_B, \ldots$ durch das folgende Produkt gegeben:

$$|K| = \prod_{X \in \{A,B,\ldots\}} (n_X \cdot o_X)^{(n_X \cdot i_X)}$$

Wenn man die Gesamtheit aller Teilzustände $(s_A, s_B, \ldots)$ als einen Gesamtzustand auffasst, dann ergeben sich $n = n_A \cdot n_B \cdot \ldots$ mögliche Zustände. Für die Anzahl der möglichen Outputs ergibt sich $o = o_A \cdot o_B \cdot \ldots$. Weil die Inputs nicht zwangsläufig für jede FSM unterschiedlich sind, ergibt sich für die Anzahl i der möglichen Inputkombinationen die Bedingung $\max(i_A, i_B, \ldots) \leq i \leq i_A \cdot i_B \cdot \ldots$. Würde man eine einzelne FSM mit dieser Anzahl an Zuständen, Inputkombinationen und Outputkombinationen implementieren und optimieren, ergäbe sich ein deutlich größerer Suchraum:

$$\left(\prod_{X \in \{A,B,\ldots\}} (n_X \cdot o_X) \right)^{i \cdot \prod_{X \in \{A,B,\ldots\}} n_X} \leq |K| \leq \left(\prod_{X \in \{A,B,\ldots\}} (n_X \cdot o_X) \right)^{\prod_{X \in \{A,B,\ldots\}} (n_X \cdot i_X)}$$

Mit jeder zusätzlichen Trennung wird der Exponent in der Gleichung für mehrere FSMs im Vergleich zur anderen Gleichung kleiner. Zunächst scheint also eine drastische Reduzierung

der Komplexität mit dieser Technik erreicht zu sein. Natürlich bedeutet das aber auch, dass viele Lösungen, die mit einer großen FSM realisiert werden können, nicht im Suchraum mit zwei oder mehr kleineren FSMs enthalten sind. Die Grundlage der Trennung war jedoch, dass die Teilaktionen $(a_A, a_B, \dots)$ unabhängig voneinander sein sollen. Und gerade diese Lösungen bleiben im kleineren Suchraum enthalten. Für das Beispiel von vorhin bedeutet das, dass z. B. die Lösungen, die bei Entscheidungen von der „Dreh-FSM“ den Zustand und die Inputs der „Flag-FSM“ berücksichtigen (oder umgekehrt), nicht im kleineren Suchraum enthalten sind.

In [EH10a] wurden Agenten für den ATAC mit zwei FSMs evolviert. Eine FSM wurde zur Steuerung der Bewegung des Agenten eingesetzt, die andere zur Steuerung von Veränderungen auf Flags in der Umwelt.

5.5.3 Time-Shuffling

Bei der Auswertung einiger Simulationsergebnisse von Agenten, die mit unterschiedlichen FSMs kontrolliert werden und auf verschiedenen Welten simuliert werden (beispielsweise in [HHB06]), konnte festgestellt werden, dass einige FSMs gute Lösungen für einen Teil der Welten darstellten, aber auf dem anderen Teil nicht gut funktionierten. Bei anderen FSMs war es genau umgekehrt. Das legt die Schlussfolgerung nahe, dass für unterschiedliche Welten unterschiedliche Strategien optimal oder notwendig sind, welche von einer in ihrer Größe beschränkten FSM eventuell nicht alle gleichzeitig realisiert werden können.

Die Idee hinter der *Time-Shuffling*-Technik (TS) ist es, den Agenten während der Simulation, einen Strategiewechsel zu ermöglichen. Z. B. in einer Situation, in der sich der Agent in einem Zyklus befindet, aus dem er mit einer FSM nicht ausbrechen kann, bietet sich ein Wechsel der Kontrollfunktion an. Zuerst wurde TS in [EHH08a] beschrieben. Das Ziel war es, zwei bereits durch Optimierung gefundene, einzelne FSMs A und B zu einer hybriden Kontrollfunktion zu kombinieren, um das Verhalten gegenüber den einzelnen FSMs zu verbessern. Die Kombination der zwei FSMs ist dabei dergestalt, dass für eine bestimmte Zeit T die FSM A aktiv ist und danach für die gleiche Zeit die FSM B. Dieser Wechsel wird dann zyklisch wiederholt. In [EHH08a] wurde zunächst mit der *Time-Shuffle-Periode* $T = 1$ experimentiert, in weiteren Arbeiten dann mit höheren Werten bis zu $T = 1536$ [EH08a]. In [EH10b] wurden zwei Time-Shuffle-Perioden T_A und T_B verwendet, die die aktive Zeit für beide FSMs unterschiedlich definieren.

Für das Time-Shuffling wurden drei verschiedene Modi entwickelt (Abb. 5.14):

- *Modus g*: gemeinsamer Zustand. Der Agent arbeitet mit beiden FSMs gleichzeitig, aber es gibt nur einen Zustand s. Beide FSMs erhalten den gleichen Input x und geben abhängig davon und von momentanen Zustand s die Outputs y_A bzw. y_B sowie die Folgezustände s'_A bzw. s'_B aus. Abhängig vom globalen Parameter $\lambda = (\lfloor g/T \rfloor \mod 2)$ ist entweder $s' = s'_A$ und $e = y_A$ oder $s' = s'_B$ und $e = y_B$. Bei zwei unterschiedlichen Time-Shuffle-Perioden wird der globale Parameter $\lambda = \lceil \lfloor (g \mod (T_A + T_B))/T_A \rfloor / (T_A + T_B) \rceil$ verwendet. g gibt die momentane Generation des CA an.

- *Modus s*: separater Zustand mit simultaner Ausführung. In diesem Modus besitzen beide FSMs einen eigenen Zustand s_A bzw. s_B. Beide FSMs werden simultan in jedem Schritt ausgeführt und aktualisieren ihren Zustand. Aber nur einer der beiden Outputs wird zur Entscheidung e. Der globale Parameter λ ist genau so definiert wie im Modus g.

- *Modus a*: separater Zustand mit alternierender Ausführung. Wie im Modus *s* gibt es je einen Zustand für beide FSMs und die Entscheidung wird abhängig von λ aus einem der beiden Outputs gewählt. Die Aktualisierung des Zustands erfolgt allerdings nicht simultan. Stattdessen wird in Abhängigkeit von λ nur der Zustand der aktiven FSM aktualisiert. Das Signal *ce* (clock enable) aktiviert die Aktualisierung des jeweiligen Zustandsregisters.

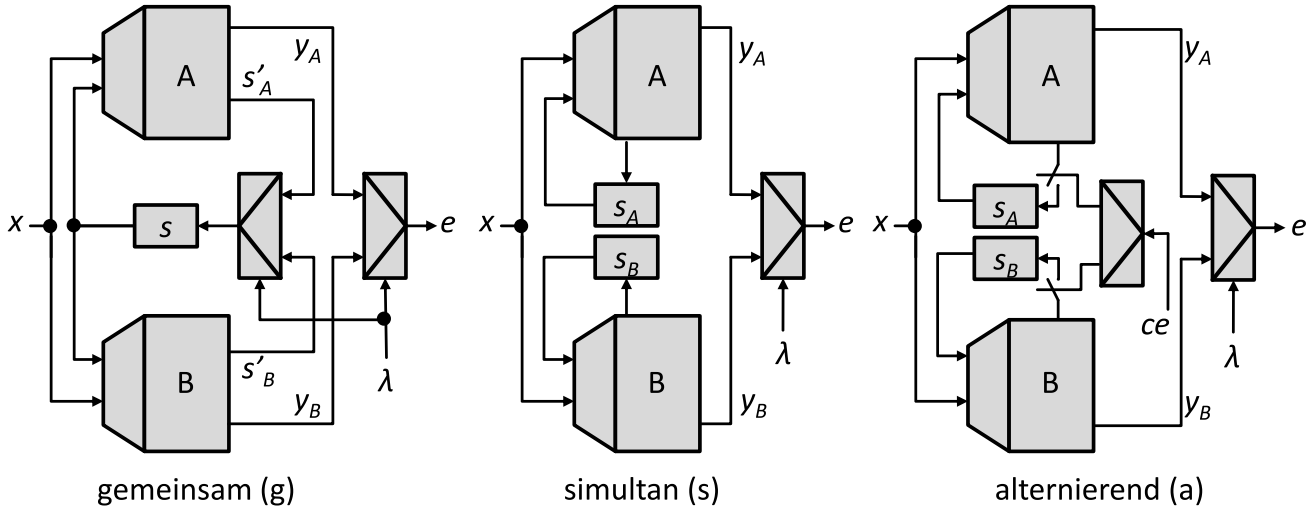

Abb. 5.14.: Die drei Time-Shuffling-Modi: gemeinsamer Zustand (g), separate Zustände mit simultaner Ausführung (s) und separate Zustände mit alternierender Ausführung (a).

Die Anwendung des TS stellt gegenüber den Grundvarianten der Beispielprobleme eine Veränderung in der Struktur der Zelle, genauer in der Struktur der Kontrollfunktion dar. Die Schnittstellen zu Sensor und Aktuator bleiben unverändert.

Die Anwendbarkeit des TS ist bei allen drei Beispielsystemen gegeben. Die heuristische Optimierungsmethode muss hier speziell angepasst werden. Schließlich müssen hier zwei FSMs, die zueinander passen, gefunden werden. Zwei Vorgehensweisen sind untersucht worden.

Separates Optimieren. Die FSMs *A* und *B* können zunächst als eigenständige Kontrollfunktion ohne TS optimiert und danach in einem der drei Modi zur hybriden Kontrollfunktion zusammengefügt werden. Die Fragen, die sich dabei stellen, sind: Sind zwei gute Einzelstrategien in Kombination noch besser? Und außerdem: Wie sollen die Einzelstrategien optimiert werden, für unterschiedliche oder für gleiche Welten? Wählt man unterschiedliche initiale Konfigurationen, so werden die einzelnen FSMs für spezielle Fälle trainiert und sind möglicherweise schlecht auf den Welten, für die sie nicht trainiert worden sind. Dies soll dann durch die Strategiewechsel kompensiert werden [EH09c]. Wählt man hingegen gleiche initiale Konfigurationen aus, erhält man zwei allgemein gute FSMs, die dennoch unterschiedliche Eigenschaften haben können. Durch TS sollen sich die positiven Eigenschaften (im Sinne der Problemlösung) ergänzen [EHH08a, EH08a].

Selbst wenn auf der einen Seite die Kompensation und auf der anderen Seite die positive Ergänzung gut funktionieren, ist noch nicht klar, wie man die besten Time-Shuffle-Perioden findet. Eine Gesamtlösung ist definiert durch das Tupel (A, B, T) bzw. das Tupel (A, B, T_A, T_B). Die Werte für T sind sinnvoll begrenzt bei einer Zahl, die die Gesamtzahl der zulässigen Generationen oder der Generationen, die eine einzelne FSM für die Problemlösung benötigt, überschreitet. Deshalb ist es möglich für ein gesetztes Paar (A, B), durch Ausprobieren zu einer Lösung zu kommen. Das gleiche gilt für die Wahl des Modus.

Gemeinsames Optimieren. Die FSMs A und B sowie die Time-Shuffle-Periode T bzw. die Time-Shuffle-Perioden T_A und T_B können auch gemeinsam optimiert werden [EH10b]. Das Genom für den GA muss entsprechend angepasst werden, so dass die hybride Kontrollfunktion direkt optimiert werden kann. Die Genome der beiden FSMs werden einfach konkateniert und dahinter ein weiteres Gen für T angefügt, oder zwei Gene für T_A und T_B (Abb. 5.15). Für hybride Kontrollfunktionen ergeben sich dann auch andere Suchräume. Sei die Anzahl der möglichen Time-Shuffle-Perioden begrenzt auf $\#T = \Pi$, dann ist im allgemeinen Fall mit zwei Time-Shuffle-Perioden die Größe des Suchraums abhängig von der Anzahl der Zustände n bzw. n_A und n_B, der Anzahl der möglichen Inputkombinationen i und Outputkombinationen o durch $|K| = \Pi^2 \cdot (n_A \cdot o)^{(n_A \cdot i)} \cdot (n_B \cdot o)^{(n_B \cdot i)}$ gegeben. Bei nur einer Time-Shuffle-Periode ergibt sich $|K| = \Pi \cdot (n_A \cdot o)^{(n_A \cdot i)} \cdot (n_B \cdot o)^{(n_B \cdot i)}$. Für den Modus g gilt $n_A = n_B$. Wenn beide FSMs identisch sind ($A = B$), dann sind die Modi g und s äquivalent zur Verwendung einer einzelnen FSM ohne TS. Im Modus a nicht.

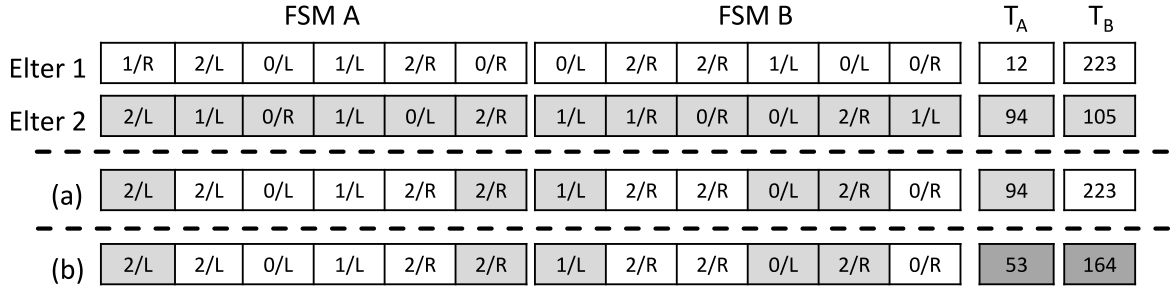

Abb. 5.15.: Beispiel für die Codierung und Rekombination für hybride Kontrollfunktionen mit TS und zwei 3-Zustands-FSMs. (a) Nachkomme mit UN und uniformer TS-Rekombination, (b) Nachkomme mit UN und TS-Rekombination mit Mittelwertbildung.

Die Rekombination dieses erweiterten Genoms kann durch die Techniken UN oder 1P separat für die beiden FSMs durchgeführt werden. Für den letzten Teil des Genoms sind zwei Rekombinationstechniken entwickelt worden. Bei der *uniformen TS-Rekombination* (Abb. 5.15(a)) wird für jede Periode entweder der Wert des ersten Elters oder der des zweiten Elters übernommen. Bei der *TS-Rekombination mit Mittelwertbildung* (Abb. 5.15(b)) werden die beiden Werte der Eltern gemittelt: $T_{Nachkomme} = (T_{Elter1} + T_{Elter2})/2$. Die Mutation kann entweder auf die FSMs beschränkt bleiben, oder die Gene der Time-Shuffle-Perioden mit einbeziehen. Diesen wird dann je nach Mutationswahrscheinlichkeit ein zufälliger zulässiger Wert zugewiesen.

Mehrere Erweiterungen und Varianten der Time-Shuffling-Technik sind denkbar. Es könnten mehr als zwei FSMs verwendet werden um die Strategienvielfalt weiter zu erhöhen oder es könnten andere globale (oder auch lokale) Parameter als die Zeit verwendet werden, um zwischen den Strategien zu wechseln um direkt auf bestimmte Situationen reagieren zu können.

5.5.4 Heterogene MAS mit verschiedenen Agententypen

Während TS einen zeitabhängigen Strategiewechsel in jedem Agenten ermöglicht, wird in diesem Abschnitt eine andere Vorgehensweise beschrieben, die ebenfalls mehrere Strategien vereint, aber nicht zeitbehaftet wechselt, sondern räumlich, bezogen auf die initiale Konfiguration einer Welt. Dahinter steckt folgende Idee. In MAS mit mehreren Agenten können Agenten mit unterschiedlichen Kontrollfunktionen (Strategien) eingesetzt werden, die jede für sich mögli-

cherweise optimal für bestimmte Welten sind. Dadurch sind in jeder Welt spezialisierte Agenten vorhanden und das Problem kann möglicherweise insgesamt besser gelöst werden als mit Agenten mit ausschließlich gleicher Kontrollfunktion. Aus Modellierungssicht sind das MAS mit Agenten unterschiedlichen Typs (heterogene MAS), deren Struktur bis auf die Kontrollfunktion gleich ist. In der Zellstruktur müssen also wie auch beim TS mehrere FSMs untergebracht sein.

Je nach Problemstellung ist die initiale räumliche Platzierung der Agenten offen oder fest vorgegeben. In beiden Fällen kann jedoch die Verteilung der Agententypen variiert werden.

Wie bei TS kann eine Optimierung separat durchgeführt werden und im Anschluss die optimierten FSMs in heterogenen MAS eingesetzt werden. Die Frage nach der Verteilung muss dann ebenfalls noch geklärt werden. Die Größe des Suchraums hängt hier natürlich von der Anzahl der Agenten und der Anzahl der verschiedenen FSMs ab. Bei wenigen Kombinationsmöglichkeiten ist gegebenenfalls das Ausprobieren aller Möglichkeiten ein probates Mittel. Alternativ dazu kann auch gleich das gesamte System mit zwei oder mehr FSMs inklusive der initialen Verteilung optimiert werden. Eine entsprechende Codierung der Verteilung der Agenten muss dafür gefunden werden, so dass sie zusammen mit den FSMs ein Genom bilden kann. Es können auch *Koevolutionäre Genetische Algorithmen* [ES03, S. 222ff.] zum Einsatz kommen. In diesem Fall muss es sich um eine kooperative Koevolution handeln, weil die verschiedenen FSMs zusammen arbeiten sollen (es gibt nur ein gemeinsames Ziel) um das Gesamtsystem zu optimieren.

In [EHH08b] wurden heterogene MAS für das CEP auf ihre Effektivität untersucht. Die FSMs wurden separat evolviert und später im heterogenen MAS mit je zwei Agententypen verwendet. Einige ausgewählte Verteilungsschemata der Agententypen und mehrere Fälle mit einer unterschiedlichen Anzahl von Agenten wurden verglichen.

5.5.5 Randomisierte FSMs

Randomisierte FSMs sind eine Möglichkeit, die Robustheit eines FSM-kontrollierten Agenten gegenüber Veränderungen der initialen Konfiguration der Welt zu steigern. Angenommen, eine FSM ist für eine bestimmte Menge von Welten optimiert worden, aber in einer weiteren Welt treten Situationen auf, die vorher nicht aufgetreten sind. Da die FSM nicht daraufhin optimiert wurde, kann es sehr leicht geschehen, dass z. B. *Deadlocks* oder *Livelocks* auftreten, also Situationen, in denen der Agent gar keine Aktion mehr ausführt, die seine Lage ändern würde, oder in denen der Agent zyklisch die gleichen Aktionen ausführt und er immer wieder in der gleichen Situation landet.

Ein anderer Grund für das Auftreten von Deadlocks und Livelocks kann die Homogenität der Agenten sein. Um das zu illustrieren, sei hier eine (initiale) Konfiguration des ARP mit vier homogenen Agenten gegeben, die alle die gleiche Kontrollfunktion haben (Abb. 5.16(a)). Die Zielpositionen aller Agenten liegen in deren Frontzelle. Das bedeutet, dass alle Agenten denselben (reduzierten) Input x_r bekommen, da das Ziel aus Agentensicht in der gleichen Zone liegt. Die FSM ist initial bei allen Agenten im Zustand $s = 0$. Also wird jeder Agent dieselbe Entscheidung e treffen. Der Weg aller Agenten ist blockiert und ein Swap ist auch nicht möglich. Alle Agenten werden sich also nur drehen können und auf ihrer Position verharren. Da sich alle Agenten in der gleichen Richtung drehen, entsteht danach eine von vier möglichen Folgekonfigurationen, auf die wiederum eine dieser vier Konfigurationen folgt. Es spielt dabei keine Rolle, was für eine FSM die Agenten kontrolliert, sondern nur, dass es jeweils die gleiche

ist. Auch in größeren Agentenwelten und solchen mit Rand lassen sich Livelocks konstruieren (Abb. 5.16(b)).

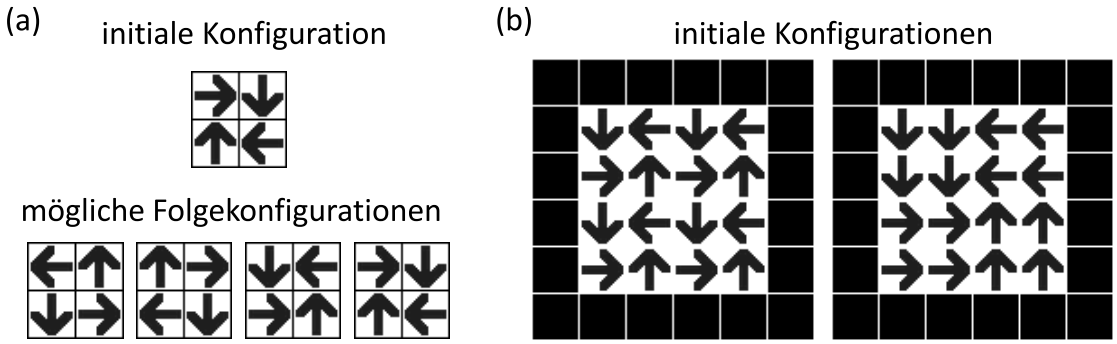

Abb. 5.16.: Livelock-Situationen im ARP mit homogenen Agenten. (a) auf einer Konfiguration mit 4 Zellen und Wrap-Around, (b) auf Konfigurationen mit 16 Zellen und Rand. Die Zielpositionen der Agenten sind in der initialen Konfiguration jeweils in der Frontzelle.

Sowohl die Deadlocks/Livelocks, die durch Homogenität verursacht sind, als auch die, die durch schlechte Optimierung entstanden sind, können durch probabilistisches Verhalten aufgelöst und damit die Robustheit der Agenten erhöht werden. In den Fällen aus Abb. 5.16 können die Situationen nur mit einem Swap aufgelöst werden, der nur zustande kommen kann, wenn die Agenten sich nicht homogen verhalten. Randomisierte FSMs zur Agentenkontrolle für das ARP wurden in [EH10c] beschrieben. In [DL06] wurden die Agenten für das CEP randomisiert.

Bei einer randomisierten FSM (Abb. 5.17) kann die Entscheidung durch zwei mögliche Weisen getroffen werden. Entweder ist der Output der FSM die Entscheidung ($e = y$) oder es wird aus allen möglichen Entscheidungen e_0 bis e_{o-1} durch eine ganzzahlige Zufallszahl $r \in \{0, 1 \ldots o-1\}$ eine komplett zufällige Entscheidung ausgewählt ($e = e_r$). Der randomisierten FSM ist eine Wahrscheinlichkeit $0 \leq p \leq 1$ zugeordnet. Ein Signal $ce(p)$ (clock enable) wird ausgewertet, um zu entscheiden, ob die zufällige Entscheidung oder der Output der FSM an den Aktuator weitergeleitet wird. In der hier vorgestellten Konstruktion wird das Signal außerdem dazu verwendet, die Aktualisierung des Zustands s zuzulassen bzw. zu verhindern. Bei Auswahl einer zufälligen Entscheidung wird demnach auch der Zustand der FSM nicht aktualisiert. Das Signal $ce(p)$ nimmt dabei mit der Wahrscheinlichkeit p den Wert 0 an (zufällige Entscheidung) und mit der Wahrscheinlichkeit $1 - p$ den Wert 1 (Entscheidung der FSM).

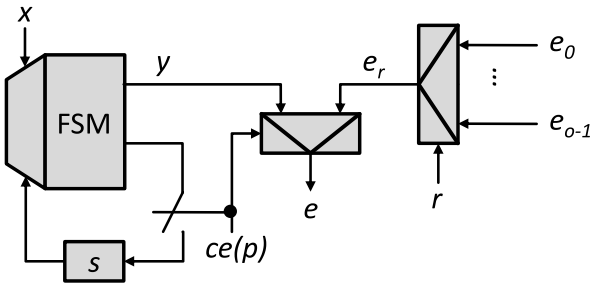

Abb. 5.17.: Randomisierte FSM zur Kontrolle des Agenten.

Durch die Einbindung von zufälligen Werten in die Kontrollfunktion, und damit in die Zellregel insgesamt, entsteht mit der Verwendung von randomisierten FSMs ein PCA (Abschn. 2.2.1).

Randomisierte FSMs können in allen Beispielsystemen eingesetzt werden, sofern keine deterministische Simulation verlangt ist. Die heuristische Optimierungsmethode muss angepasst werden. Dies geschieht in ganz ähnlicher Weise wie beim TS. Im Genom wird der Wert p an das Ende des Strings gehängt. Die Genauigkeit dieses Wertes muss beschränkt sein, wenn man den Suchraum beschränken will (z. B. auf Zehntel Prozent). Weiterhin kann man das Intervall begrenzen. Sehr hohe Werte sind z. B. oft nicht sinnvoll, weil dann die FSM nur selten benutzt wird und damit deren Strategie hinfällig wird. Die Größe des Suchraums lässt sich dann in Abhängigkeit der Anzahl der möglichen Wahrscheinlichkeitswerte $\#p = \Pi$ angeben: $|K| = \Pi \cdot (n \cdot o)^{(n \cdot i)}$. Wie beim TS kann die Rekombination der FSM mit den Techniken UN oder 1P realisiert werden. Die Wahrscheinlichkeit p kann entweder uniform oder mit Mittelwertbildung rekombiniert werden.

Eine Schwierigkeit beim Evolvieren von randomisierten FSMs mit GA liegt in der Bestimmung der Qualität einer Lösung. Jede Lösung ist zufallsbehaftet, deshalb kann ein durch Simulation ermittelter Fitnesswert nicht immer den gleichen Wert annehmen. Zur Bestimmung des Fitnesswerts innerhalb der Iterationen des GA sollten deshalb mehrere Simulationen durchgeführt werden, um die Gefahr des Aussortierens eines eigentlich (im Mittel) guten Kandidaten zu verringern. Ganz auszuschließen ist diese Gefahr jedoch nicht. Da außerdem die Simulationszeit erhöht wird, kann es in manchen Fällen eine bessere Strategie sein, zunächst FSMs ohne Zufall zu evolvieren und im Nachhinein zu randomisieren, um einen positiven Effekt zu erzielen.

5.6 Kapitelzusammenfassung

Grundsätzliche Fragen zur Optimierung von Agentenverhalten in emergenten Systemen wurden diskutiert und Heuristiken vorgestellt, die für die Optimierung der Kontrollfunktion geeignet sein können.

Genetische Algorithmen können zur Optimierung von FSM-kontrollierten Agenten, wie in Abschn. 5.4 und Abschn. 5.4.3 beschrieben, eingesetzt werden, die Parameter und Techniken zur Mutation und Rekombination können variiert werden. Die Ergebnisse waren im Experiment zufriedenstellend und obwohl die Suche gegenüber zufälligen oder exakten Aufzählungsverfahren deutlich schneller ist, wird durch die notwendige Simulation immer noch viel Rechenzeit verbraucht.

Einige der in Abschn. 5.5 beschriebenenen Techniken zielen darauf ab, den GA durch manuelles Eingreifen in die Modellierung der Kontrolleinheit zu entlasten bzw. bei gleichem Aufwand bessere Lösungen zu finden. Durch die Verwendung von mehreren *separaten FSMs*, *Time-Shuffling*, MAS mit *heterogenen Agenten* und *randomisierten FSMs* sollen bestimmte negative Effekte wie Spezialisierung auf bestimmte Probleme (Instabilität gegenüber Veränderungen der Umwelt), Deadlocks oder unnötige Verknüpfungen von unabhängigen Aktionen reduziert werden.

Die Möglichkeit der Anwendung der manuellen Techniken ist abhängig von der Problemstellung. Die heuristischen Verfahren und manuellen Techniken werden in Kap. 6 mit den Beispielsystemen aus Kap. 4 eingesetzt und auf Effektivität überprüft.

6 Einsatz und Validierung der Techniken auf den Beispielsystemen

In diesem Kapitel werden die in Kap. 5 entwickelten Techniken auf die Beispielsysteme aus Kap. 4 angewendet und evaluiert. Die *Effektivität* (Wird ein Agentenverhalten mit hoher Fitness erzeugt?) und *Effizienz* (Sind die Kosten, z. B. die Berechnungszeit, für das Erzeugen des guten Agentenverhaltens niedrig?) der Techniken soll so für bestimmte Beispiele validiert werden. In Kombination mit den vorher vorgestellten manuellen Optimierungstechniken stellt dies die Komplettierung der Realisierung des zweiten Teilziels der Arbeit dar.

Einige der Techniken wurden auf den Beispielsystemen auch in Kombination getestet und die Ergebnisse in einer Reihe von Veröffentlichungen präsentiert (Tab. 6.1). In den ersten vier Spalten ist angegeben, ob in der jeweiligen Untersuchung eine Optimierung durch Maßnahmen der Anpassung der Agentenwelt stattfand. Die nächsten sechs Spalten beinhalten Techniken, die die Kapazitäten der Kontrollfunktion und die Modellierung der Kontrolleinheit anpassen. In den letzten vier Spalten ist angegeben, ob und welche heuristischen Verfahren eingesetzt wurden. Zusätzlich ist angegeben, in welchen Abschnitten dieser Arbeit, die Beschreibung der jeweiligen Experimente zu finden sind.

Tab. 6.1.: Veröffentlichte Untersuchungen zu den Optimierungstechniken.

	Maßnahmen zur Anpassung der Welt				Techniken zur Optimierung der Agentenkontrolle						heuristische Suchverfahren					
Beispielsystem	**Anzahl Agenten k**	**Wrap/Rand**	**Weltgröße**	**Topologie**	**Agentenfähigkeiten**	**Anzahl Zustände n**	**separate FSMs**	**Time-Shuffling**	**heterogene MAS**	**randomisierte FSMs**	**GA**	**Inselmodell GA**	**inkrementeller GA**	**GP**	**Veröffentlichungen**	**Abschnitt**
CEP											•				[EHG11]	5.4
CEP						•					•	•			[EHH09]	5.4.3
CEP														•	[KEFH09]	
CEP	•								•						[EHH08b]	6.1.1
CEP	•							•							[EHH08a, EH08a]	6.1.2
ATAC		•									•	•			[EH08b, HE08]	
ATAC					•						•	•			[EH09d]	6.2.2
ATAC		•						•			•	•			[EH09c, EH10b]	6.2.1
ATAC							•				•	•			[EH10a]	6.2.2
ARP	•		•		•						•	•			[EH09b]	
ARP	•		•								•	•			[EH10d]	6.3.1
ARP										•	•	•			[EH10c]	6.3.2
ARP				•							•	•	•		[EHD10]	

Grundsätzlich können für jedes Problem alle Techniken und Maßnahmen gleichzeitig angewandt und alle Parameter gleichzeitig angepasst werden. Allerdings steigt dadurch natürlich die Komplexität für die Heuristik beträchtlich, da sich die Anzahl aller Kombinationsmöglichkeiten durch Potenzierung ergibt. Außerdem steigt der Aufwand für die Realisierung einer Zelle und damit des ganzen CA, wenn die Kontrollfunktion mit mehreren kombinierten Techniken erweitert wird, die ja unter anderem die Anzahl und die Größe der FSMs erhöhen.

Wenn nur wenige Techniken kombiniert werden, wird die Interpretation der Ergebnisse in der Hinsicht einfacher, dass sich nicht die Effekte mehrerer verwendeter Techniken überlagern, sich gegenseitig aufheben usw. Die Ergebnisse der Optimierungen sind dann aber zunächst natürlich auch immer nur für den getesteten Spezialfall gültig. Dennoch können sie in einigen Fällen einen Aufschluss darüber geben, ob eine bestimmte Technik tendenziell eine Verbesserung im Agentenverhalten bringen könnte und ob die gewählte Heuristik für die durch die manuelle Technik entstandene Struktur der Kontrolleinheit geeignet ist.

Aufgrund der großen Anzahl wird in dieser Arbeit nur eine Auswahl an Untersuchungen und Ergebnissen zu den einzelnen Techniken präsentiert. In Abschn. 6.1 werden zuerst Techniken auf das CEP angewendet, dann auf den ATAC in Abschn. 6.2 und schließlich auf das ARP in Abschn. 6.3. In Abschn. 6.4 werden weitere Ergebnisse präsentiert, ohne die jeweiligen Untersuchungen detailliert zu beschreiben.

6.1 Einsatz von heterogenen Agententypen und TS auf das Creatures' Exploration Problem

In diesem Abschnitt werden zwei Experimente beschrieben, die auf dem CEP ausgeführt wurden. In einem Experiment wurde die Effektivität des Einsatzes heterogener Agenten bei gleichzeitiger Variation der Anzahl der Agenten untersucht (Abschn. 6.1.1). In dem anderen Experiment wurde die Effektivität von Time-Shuffling sowie der Variation der Time-Shuffle-Periode und der Anzahl der Agenten untersucht (Abschn. 6.1.2).

6.1.1 Experiment mit heterogenen MAS und Variation der Agentenanzahl

Mit dem im Folgenden beschriebenen Experiment soll herausgefunden werden, ob der Einsatz von heterogenen MAS eine Steigerung der Effizienz gegenüber MAS mit nur einem Agententypen bringen kann und wie diese im Zusammenhang mit der Anzahl der Agenten steht. Der Effizienzbegriff ist hier auf die Agenten und deren Problemlösung bezogen, nicht auf die Technik der Optimierung des Agentenverhaltens. Kann die Technik Agenten mit hoher Effizienz und Effektivität hervorbringen, so ist sie effektiv. Effizient ist die Technik dann, wenn der Aufwand für das Finden dieses effektiven und effizienten Agentenverhaltens wenig Ressourcen verbraucht. Die Ergebnisse dieses Experiments wurden teilweise in [EHH08b] veröffentlicht. Eine andere Untersuchung über die Anzahl der Agenten, aber ohne heterogene Systeme wurde in [HH06b] durchgeführt.

Das Problem der Agenten ist die Grundvariante des CEP. Es werden Welten der Größe 33×33 Zellen mit Rand und 129 weiteren Hindernissen verwendet. Die Anordnung der Hindernisse ist wie in den vier symmetrischen Konfigurationen ($j = 1 \ldots 4$) in Abb. 5.11 (außer links oben) und den 12 Konfigurationen in Abb. B.3 (S. 167), davon sechs manuell erzeugte ($j = 5 \ldots 10$)

und sechs mit zufällig verteilten Hindernissen ($j = 11 \ldots 16$). Zu jeder dieser Anordnungen ist je eine initiale Konfiguration mit $k \in \{1, 2, 4, 8, 12, 16, 28, 32, 60, 64\}$ Agenten gegeben. Die Agenten sind gleichmäßig am Rand mit Blickrichtung nach innen auf den initialen Konfigurationen angeordnet (Abb. B.4, S. 168). Die heterogenen MAS sind in diesem Experiment immer nur aus zwei verschiedenen Agententypen (mit FSM X bzw. FSM Y) zusammengesetzt. Drei verschiedene Platzierungsmodi wurden verwendet (Abb. 6.1):

- Platzierungsmodus a: Agenten mit FSM X und Agenten mit FSM Y werden abwechselnd platziert.
- Platzierungsmodus g: Agenten mit gleicher FSM werden jeweils auf gegenüberliegenden Seiten platziert. FSM X oben und unten, FSM Y links und rechts.
- Platzierungsmodus n: Agenten mit gleicher FSM werden auf zwei Seiten nebeneinander um die gleiche Ecke herum platziert. FSM X oben und rechts, FSM Y links und unten.

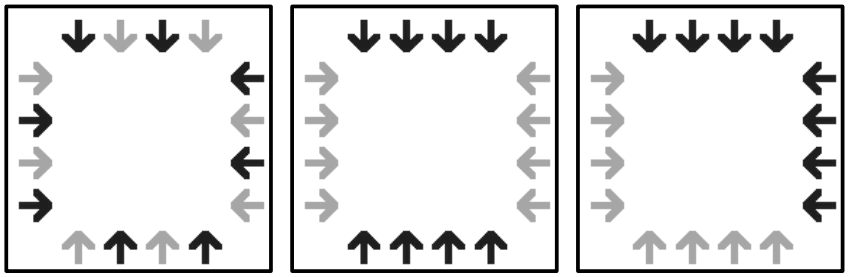

Abb. 6.1.: Platzierungsreihenfolge der heterogenen Agenten. Agenten mit verschiedener FSM sind farblich verschieden dargestellt.

Aus den Untersuchungsergebnissen von Halbach [Hal08] wurden zehn FSMs mit sechs Zuständen entnommen. Diese zehn FSMs wiesen eine gute Fitness bei mehreren Experimenten des CEP auf und sollen nun in den heterogenen Systemen verwendet werden. Die zehn FSMs werden mit den Buchstaben A bis J gekennzeichnet (Tab. 6.2). Zur Vereinfachung der Lesbarkeit, sind Trennstriche in das Genom eingefügt worden, die unterschiedliche Inputs trennen.

Tab. 6.2.: Liste der zehn FSMs mit sechs Zuständen für das Experiment mit heterogenen MAS.

FSM	Genom
A	0/R 2/R 3/R 4/L 5/L 1/L – 1/R 5/R 4/R 0/L 2/L 3/L
B	1/R 2/R 0/R 4/L 5/L 3/L – 3/R 1/L 5/R 0/L 4/R 2/L
C	1/R 2/R 0/R 4/L 5/L 3/L – 3/R 4/R 2/L 0/L 1/L 5/R
D	1/R 2/R 3/R 1/L 5/L 1/L – 1/R 0/L 2/L 4/R 3/L 1/L
E	1/R 2/L 0/R 4/L 5/L 3/L – 3/R 4/R 5/R 0/L 1/L 2/R
F	1/R 2/L 0/L 4/R 5/R 3/R – 3/L 4/L 5/L 0/R 1/L 2/R
G	1/L 2/L 0/L 4/R 5/R 3/R – 3/L 1/R 5/L 0/R 4/L 2/R
H	1/L 2/L 3/R 4/L 2/R 0/L – 2/L 4/L 0/R 3/L 5/L 4/R
I	1/L 2/L 3/L 4/L 2/R 0/L – 2/L 4/L 0/R 3/R 5/L 4/R
J	1/R 2/R 3/R 0/R 4/L 5/L – 4/R 5/R 3/L 2/L 0/L 1/L

Die Bewertung der Qualität einer FSM oder heterogenen Kombinationen von FSMs, d. h. eines MAS bzw. heterogenen MAS, welches Agenten enthält, die mit diesen FSMs kontrolliert werden,

erfolgt einzeln für jede Anzahl von Agenten. FSMs und Paare von FSMs kombiniert mit einer Anzahl k werden im Folgenden als *System* bezeichnet und als X-k (mit FSM X) oder als XY_M-k (mit FSMs X und Y und Platzierungsmodus M) notiert. Es werden zur Bewertung drei Größen herangezogen:

1. die *Erfolge* $E = \sum_{j=1}^{16} suc_j$, d. h. die Anzahl der Welten, die komplett gelöst werden konnten. Eine Welt j gilt als *erfolgreich* gelöst ($suc_j = 1$), wenn alle Zellen mindestens einmal besucht worden sind. Die Simulation wurde jeweils nach 10.000 Generationen des CA abgebrochen, falls die Welt nicht vorher gelöst werden konnte. Ein System mit $E = 16$ wird als *komplett erfolgreich* bezeichnet.

2. die *Abdeckung* $A = (\sum_{j=1}^{16} v_j)/(\sum_{j=1}^{16} Z_j)$, d. h. der Anteil der Zellen, die von mindestens einem Agenten besucht wurden (v_j), von der Gesamtzahl aller freien (zu besuchenden) Zellen (Z_j). In diesem Experiment gilt $Z_j = 960$, $\forall j \in \{1, \ldots, 16\}$.

3. die *mittlere Dauer* $G = (\sum_{j=1}^{16} \breve{g}_j)/16$, d. h. die Summe der Generationen $\breve{g}_j$, die benötigt wurden, um jeweils eine Welt zu lösen. Die mittlere Dauer ist nur für $E = 16$ definiert.

Die drei Größen sollen als Dominanzrelation verstanden werden, d. h. nur wenn die Erfolge gleich sind, wird die Abdeckung, und bei gleicher Abdeckung die Dauer herangezogen, um einen Vergleich der Qualität von verschiedenen Kombinationen von FSMs zu ermöglichen.

Zunächst sind die ausgewählten FSMs in MAS mit nur einem Agenten eingesetzt worden. Drei von ihnen konnten nicht eine einzige der 16 Welten lösen und insgesamt weniger als die Hälfte der Zellen besuchen (Tab. 6.3). Die besten Ergebnisse erreichten die FSMs B, G und J, die jeweils 7 von 16 Welten lösen konnten und insgesamt über 85% aller Zellen besucht haben. Auffällig ist auch, dass die symmetrischen Welten ($j = 1 \ldots 4$) für diese FSMs offenbar schwieriger zu lösen waren.

Tab. 6.3.: Erfolge suc_j, Anzahl der Erfolge E und Abdeckung A der Systeme mit einem Agenten.

FSM	suc_j																E	A
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**		
B					•		•	•	•			•		•	•		7	88,74%
G					•		•	•	•			•		•	•		7	87,32%
J						•		•		•	•		•	•	•		7	85,89%
C	•				•		•	•						•	•		6	86,92%
E						•		•		•				•			4	83,42%
A			•	•												•	3	82,38%
F										•				•			2	73,08%
D																	0	47,55%
H																	0	36,20%
I																	0	36,20%

Es ist nicht verwunderlich, dass beim Einsatz von mehr Agenten, auch die Anzahl der Erfolge steigt (Tab. 6.4). FSM J ist als einzige mit nur acht Agenten auf allen 16 Welten erfolgreich. Nur FSM A kann mit keiner Anzahl k alle Welten erfolgreich lösen. In den Fällen B und G ist die Anzahl der Erfolge nicht monoton steigend mit der Anzahl der eingesetzten Agenten. Daraus kann man schließen, dass zusätzliche Agenten nicht immer nützlich sind. In der Tendenz wird dadurch zwar die Lösung des Problems erleichtert, aber die Agenten fungieren auch als

Hindernis für die anderen Agenten und können so den Weg versperren und dadurch sogar Livelocks verursachen.

Tab. 6.4.: Anzahl der Erfolge und Dauer der Systeme mit mehreren Agenten gleichen Typs.

k	A	B	C	D	E	F	G	H	I	J
1	3	7	6	0	4	2	7	0	0	7
2	12	11	12	0	8	9	11	1	1	12
4	13	13	13	1	14	13	13	2	2	14
8	14	14	15	8	14	15	14	8	11	16/1807
12	15	16/1918	16/1556	12	15	15	15	13	15	16/1069
16	15	16/1071	16/1312	14	16/1066	15	16/984	14	15	16/768
28	15	16/456	16/475	16/1070	16/979	15	15	15	16/1415	16/463
32	15	15	16/484	16/1086	16/764	16/706	16/468	16/1779	16/932	16/382
60	15	16/276	16/257	16/430	16/389	16/486	16/245	16/591	16/928	16/224
64	15	16/261	16/285	16/416	16/393	16/382	16/314	16/627	16/705	16/257

Da mehrere Fälle komplett erfolgreich sind, können weder die Anzahl der Erfolge noch die Abdeckung als Vergleichskriterium dienen. Deshalb wird die mittlere Dauer verglichen. Die niedrigste Dauer erreicht System J-60 ($G = 224$). Erst der neuntbeste Wert wird von einem System mit weniger als 60 Agenten erreicht (System J-32 mit $G = 382$). Die höchste Dauer mit $G = 1918$ wird von System B-12 erreicht. Insgesamt schneidet FSM J bei Erfolgen und Dauer am besten ab. Durch die Erhöhung der Anzahl der Agenten kann man die Effektivität (auf die Agenten bezogen) des Systems tendenziell steigern (die Anzahl der Erfolge steigt und die Dauer verringert sich).

Systeme mit vielen Agenten haben bei der mittleren Dauer gegenüber Systemen mit wenigen Agenten demnach einen Vorteil. Schließlich ist die Anzahl der in jeder Generation theoretisch besuchbaren Zellen gleich der Anzahl der Agenten. Dennoch ist nicht klar, ob sich die Dauer antiproportional zur Anzahl der Agenten verhält. Dieses Verhältnis stellt eine vierte Vergleichsgröße dar:

4. die *mittleren Kosten* $C = k \cdot (\sum_{j=1}^{16} \check{g}_j)/(\sum_{j=1}^{16} Z_j)$, d. h. die mittlere Dauer durch die Anzahl aller Zellen, normiert mit der Anzahl der Agenten. Die mittleren Kosten geben an, wie viele Generationen im Mittel von jedem Agenten benötigt werden, um eine noch nicht besuchte Zelle zu besuchen.

Die mittleren Kosten werden nicht berücksichtigt, wenn nur die absolute Leistung des gesamten Systems gemessen werden soll. Sie können aber verwendet werden um die Effizienz (wieder auf das Agentenverhalten bezogen) und Synergieeffekte zu bestimmen. Weil die Kosten den Aufwand zum Erreichen des Ziels angeben, können sie als Effizienzwert angesehen werden. Wenn man davon ausgeht, dass der Einsatz von Agenten zur Problemlösung Kosten verursacht, so wäre es interessant zu erfahren, welches System, weniger Kosten verursacht. Angenommen, jeder Agent kostet pro Generation den Betrag 1. Die mittleren Kosten können dann als Betrag interpretiert werden, der pro besuchter Zelle investiert werden muss. Hier erreicht des beste System einen Wert von 12,72 (J-32). Im Vergleich dazu erreicht das System mit gleicher FSM und 64 Agenten nur den Wert 17,15, obwohl es absolut gesehen schneller ist. Das bedeutet, dass die Synergieeffekte im System mit 32 Agenten stärker sind. Die weiteren jeweils effizientesten Systeme jeder FSM sind: B-28 ($C = 13,30$), C-28 ($C = 13,85$), D-60 ($C = 26,89$), E-16 ($C =$

17,77), F-32 ($C = 23,54$), G-60 ($C = 15,34$), H-60 ($C = 36,94$) und I-32 ($C = 31,06$). Es scheint also keine allgemein beste Anzahl an Agenten zu geben, sondern nur abhängig von der FSM, wobei die Tendenz dennoch zu Werten von $k \approx 30$ geht. Kleinere Werte sind seltener erfolgreich oder nicht effizient, größere Werte sind ebenfalls seltener effizient.

Für die Untersuchung von heterogenen Systemen wurden alle Paare von FSMs mit allen Platzierungsmodi kombiniert und getestet. Für jeden Modus existieren 100 Möglichkeiten, die FSMs zu kombinieren, da die Reihenfolge eine Rolle spielt und die homogenen Systeme eingeschlossen sind. Tab. 6.5(a) gibt für jedes k und jeden Platzierungsmodus die Anzahl der Kombinationen an, die komplett erfolgreich waren. Im Vergleich zu den nicht heterogenen Systemen stellt sich heraus, dass bereits Systeme mit nur zwei Agenten komplett erfolgreich sein können. Das System CA_a-2 $= CA_n$-2 (Die Modi a und n sind für zwei Agenten identisch) ist das beste der drei erfolgreichen Systeme mit zwei Agenten. Es hat eine mittlere Dauer von $G = 7105$ und mittlere Kosten von $C = 14,8$. Der Platzierungsmodus spielt bei einer hohen Anzahl von Agenten keine große Rolle mehr, bei wenigen Agenten ist Modus n erfolgreicher.

Tab. 6.5.: (a) Anzahl der komplett erfolgreichen heterogenen Systeme für alle Platzierungsmodi und Agentenanzahlen, (b) schnellste (Dauer G) und (c) effizienteste (Kosten C) heterogene Systeme.

(a)

	Modus		
k	a	n	g
2	3	3	-
4	19	27	19
8	22	71	51
12	76	85	73
16	65	91	91
28	93	94	94
32	91	95	95
60	97	97	96
64	97	97	96

(b)

k	X	Y	**Modus**	G
64	B	J	a	193
60	C	J	n	203
64	J	C	n	210
64	B	J	n	211
64	J	C	g	212
64	C	J	n	213
64	C	B	g	213
64	G	J	g	214
64	C	J	g	218
60	B	C	n	220

(c)

k	X	Y	**Modus**	C
32	C	G	a	11,31
28	J	G	g	12,10
16	J	B	g	12,30
32	J	G	a	12,37
8	J	C	g	12,38
28	J	B	a	12,64
28	C	J	g	12,66
60	C	J	n	12,68
16	B	J	n	12,68
32	J	J	-	12,72

Die jeweils zehn schnellsten und effizientesten Systeme sind in Tab. 6.5(b) und (c) aufgeführt. Das System BJ_a-64 ist das schnellste. Von den 25 schnellsten Systemen ist nur ein einziges Mal die FSM F (im System FC_a-64 auf Rang 24) vertreten. Alle anderen Kombinationen bestehen ausschließlich aus den FSMs B, C, G und J. Diese vier sind auch die erfolgreichsten FSMs in Systemen mit nur einem Agenten (Tab. 6.3).

Das effizienteste System ist CG_a-32 mit mittleren Kosten von $C = 11,31$. Im Vergleich zum besten System mit nur zwei Agenten (CG_a-2) ist dies eine Abnahme der Kosten um ca. 30%. Gegenüber dem besten (erfolgreichsten) nicht heterogenen System mit acht Agenten J-8 ($C = 15,06$) sind es etwa 33%. Diese zwei Systeme werden hier als Referenz verwendet, weil sie diejenigen komplett erfolgreichen Systeme sind, die die wenigsten Agenten benötigen. Die Steigerungsraten können so als bestmögliche Synergieeffekte interpretiert werden. Wie bei der Reihenfolge der schnellsten Systeme, gibt es auch bei den effizientesten Systemen nur eines unter den ersten 25, das nicht aus den vier FSMs B, C, G und J besteht. Das effizienteste nicht heterogene System ist J-32. Nur neun heterogene Systeme können eine bessere Effizienz aufweisen.

Ergebnisse. Für heterogene Systeme mit vielen Agenten scheinen Kombinationen von FSMs, die schon für Systeme mit einem Agenten effektiv waren, geeignet zu sein. Synergieeffekte

durch die Erhöhung der Anzahl der Agenten sind möglich und treten häufig auf, allerdings nicht monoton steigend mit der Anzahl der Agenten. Die Kombination von zwei verschiedenen FSMs in einem MAS kann eine Verbesserung gegenüber Systemen mit nur einem Agententypen bedeuten (bei gleicher Anzahl der Agenten). Allerdings ist die Verbesserung gegenüber nicht heterogenen Systemen nur bei Systemen mit wenigen Agenten deutlich spürbar.

6.1.2 Experiment mit Time-Shuffling und Variation der Anzahl der Agenten

In dem nun beschriebenen Experiment geht es darum, die Effektivität der Time-Shuffling-Technik und der gleichzeitigen Variation der Anzahl der Agenten zu evaluieren. Teile der Untersuchung wurden in [EHH08a] und [EH08a] veröffentlicht.

Wie im letzten Abschnitt ist auch hier das Problem die Grundvariante des CEP. Die initialen Konfigurationen der Welten sind die gleichen 16 (Abb. 5.11 und B.3) mit entsprechender Anzahl an Agenten ($k \in \{1, 2, 4, 8, 12, 16, 28, 32, 60, 64\}$) wie in Abb. B.4 verteilt. Es wurden alle drei Time-Shuffling-Modi (Abschn. 5.5.3) eingesetzt und es wurden die FSMs aus Tab. 6.2 verwendet. Die Bewertung der Qualität einer FSM oder einer Kontrollfunktion mit zwei FSMs und Time-Shuffling entspricht der Bewertung aus dem letzten Abschnitt (Erfolge, Abdeckung, mittlere Dauer und mittlere Kosten). Zum Vergleich werden auch hier wieder die Systeme ohne TS herangezogen. Deren Ergebnisse sind bereits im letzten Abschnitt präsentiert worden (Tab. 6.3 und 6.4).

Alle Paare der FSMs A bis J wurden untereinander und mit jedem der drei TS-Modi (g, s und a) kombiniert. Ein System mit Time-Shuffling, den FSMs X und Y, des Modus M, der Anzahl der Agenten k und der TS-Periode T wird im Folgenden auch als XY_M-$k(T)$ notiert.

Zunächst wird die TS-Periode auf $T = 1$ und die Anzahl der Agenten ebenfalls auf $k = 1$ beschränkt. Für jeden Modus sind in Tab. 6.6 die besten drei Systeme mit Erfolgen E und Abdeckung A aufgelistet. Gegenüber den Systemen ohne TS ($E \leq 7$) gibt es in allen drei Modi eine Verbesserung ($E \geq 8$). Besonders deutlich ist die Verbesserung im Modus a.

Tab. 6.6.: Erfolge E und Abdeckung A der erfolgreichsten Systeme mit einem Agenten, TS und $T = 1$.

Modus	X	Y	E	A
g	G	F	8	85,44%
	E	C	8	82,50%
	C	B	8	71,50%

Modus	X	Y	E	A
s	A	G	11	96,33%
	J	B	9	92,42%
	B	J	9	85,61%

Modus	X	Y	E	A
a	B	J	12	85,84%
	B	A	11	99,38%
	C	A	11	95,24%

Tab. 6.7(a) gibt für jeden Wert k und jeden TS-Modus die Anzahl der Kombinationen an, die komplett erfolgreich waren. Berücksichtigt wurden hierbei lediglich Systeme mit $X \neq Y$, um die Vergleichbarkeit aller Modi zu gewährleisten. Systeme mit $X = Y$ sind in den Modi g und s identisch zu den Systemen ohne TS. Es gibt also nur 90 mögliche Kombinationen. Genau wie auch bei Systemen ohne TS, müssen mindestens acht Agenten eingesetzt werden, um komplett erfolgreich zu sein. Mit steigender Anzahl von Agenten steigt auch die Erfolgsrate, allerdings ist immer nur maximal ein Viertel der Kombinationen komplett erfolgreich. Bei Systemen ohne TS und 60 oder mehr Agenten sind es immerhin neun von zehn FSMs. Modus a hat leicht höhere Erfolgsraten als die anderen beiden Modi.

Tab. 6.7.: (a) Anzahl der komplett erfolgreichen Systeme aller TS-Modi und $T = 1$, (b) schnellste (Dauer G) und (c) effizienteste (Kosten C) Systeme mit TS und $T = 1$.

(a)

	Modus		
k	g	s	a
1/2	0	0	0
4	0	0	0
8	1	3	5
12	1	5	3
16	2	7	7
28	6	8	11
32	7	6	9
60	12	14	17
64	13	20	21

(b)

k	X	Y	Modus	G
64	B	C	g	218
60	J	J	-	224
60	B	C	g	233
60	G	G	-	245
64	J	J	-	257
60	C	C	-	257
64	B	B	-	261
60	B	B	-	276
64	C	C	-	285
64	G	G	-	314

(c)

k	X	Y	Modus	C
32	J	J	-	12,72
28	B	C	g	12,74
16	J	B	-	12,80
28	B	B	-	13,30
12	J	J	-	13,36
28	J	J	-	13,49
28	C	C	-	13,85
60	J	J	-	14,03
64	B	C	g	14,51
60	B	C	g	14,58

Die zehn absolut schnellsten Systeme sind in Tab. 6.7(b) aufgelistet. Das schnellste System ist BC_g-64(1). Acht der schnellsten zehn Systeme sind allerdings Systeme ohne TS. Das effizienteste System ist J-32 ohne TS, sowie sechs weitere der effizientesten zehn Systeme sind ohne TS. Das effizienteste System mit TS ist BC_g-28(1).

Der Einsatz von TS bringt also bei einer TS-Periode $T = 1$ nur dann Vorteile, wenn wenige Agenten eingesetzt werden. Einer der möglichen Gründe für das schlechte Abschneiden der TS-Technik kann durch die Funktionsweise der FSM erklärt werden. Abb. 6.2 zeigt die FSM J als Zustandsübergangsdiagramm und dessen Zyklen für gleiche Inputs getrennt dargestellt. Für die anderen FSMs aus dieser durch Aufzählung gefundenen Auswahl gilt wie für J, dass in der Regel nur Zyklen auftreten. Ein Zustand, der nicht erreicht wird (bei gleichem Input) ist die Ausnahme. Diese Zyklen induzieren Bewegungsmuster, die offenbar den Erfolg dieser FSMs ausmachen. In Fall J sind die Zyklen für den konstanten Input $m = 1$ (d. h. der Agent gerät in keinen Konflikt):

- ein Weg in der Diagonalen, hervorgerufen durch die zyklische Ausführung der Aktionen Rm und Lm,
- eine zyklische Bewegung im Quadrat auf denselben vier Zellen durch die wiederholte Ausführung von Lm.

Andere FSMs besitzen auch Zyklen der Länge drei oder vier, durch die sich Bewegungen in der Horizontalen oder Vertikalen auf einer Breite von zwei Zellen ergeben können, oder zyklische Bewegungsmuster auf mehr als vier Zellen. Die Zyklen für den Input $m = 0$ bedeuten, dass sich der Agent solange er sich nicht nach vorne bewegen kann, immer in die gleiche Richtung dreht. Auch das ist verstärkt in den anderen FSMs wiederzufinden. Wenn nun in jeder Generation, also nach jedem Zustandsübergang, die aktive FSM gewechselt wird, dann werden diese Zyklen aufgetrennt und das ursprünglich gute Verhalten ist nicht mehr gegeben. Deshalb scheint es sinnvoller, den Agenten mit TS eine höhere TS-Periode zu geben, so dass für einen gewissen Zeitraum ein Verhalten, das ja vorher als gut beurteilt wurde, aufrecht erhalten bleibt.

Zwei Fragen stellen sich als Folge der Erhöhung der TS-Periode. Erstens, was ist die optimale Periode für welche Anzahl der Agenten? Und zweitens, welches ist der beste TS-Modus? Um der Antwort darauf näher zu kommen, wurden weitere Simulationen durchgeführt: alle Kombinationen von FSMs und TS-Modi und den TS-Perioden $T \in \{1, 3, 5, 6, 12, 24, 48, 96, 384, 1536\}$.

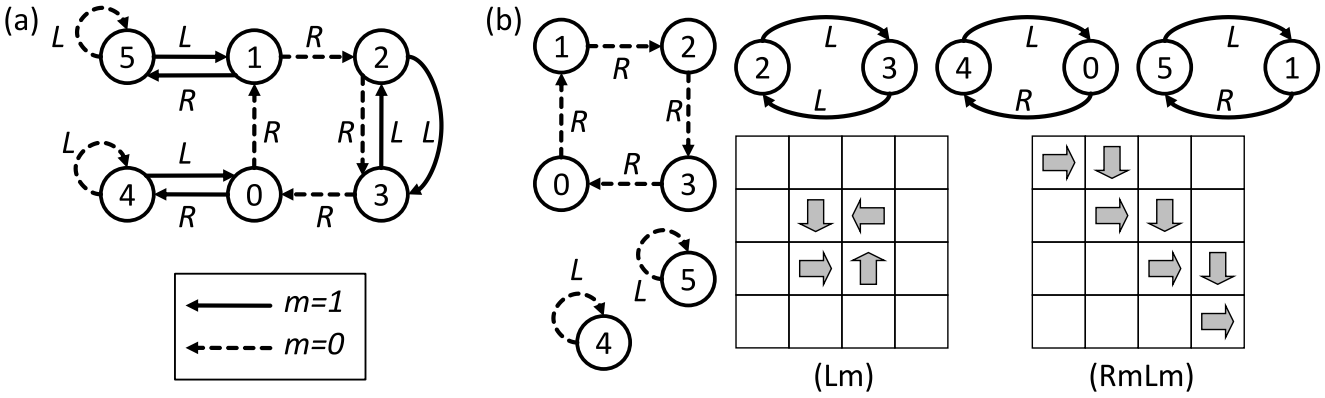

Abb. 6.2.: (a) FSM J als Zustandsübergangsgraph und (b) dessen Zyklen und deren induzierte Bewegungsmuster.

In Abb. 6.3 sind die mittleren Erfolgsquoten der unterschiedlichen Modi in Abhängigkeit der TS-Periode abgetragen. Es wird dabei unterschieden zwischen den drei Modi mit der Bedingung $X \neq Y$, einem vierten Fall mit Modus a und $X = Y$, sowie Systemen ohne TS. Die mittlere Erfolgsquote ist der Anteil der komplett erfolgreichen Systeme für einen Wert T von allen 900 möglichen Systemen mit diesem Wert T (90 Kombinationen von FSMs, 10 Werte von k). 41 von 100 Systemen ohne TS (10 FSMs, 10 Werte von k) sind komplett erfolgreich. Alle TS-Modi erreichen einen deutlich höheren Anteil als 41% für $T > 6$ oder $T > 12$. Der Modus a weist leicht höhere Werte auf, als die anderen Modi. Die Kurvenverläufe deuten an, dass es, unabhängig vom Modus, eine optimale TS-Periode gibt, die bei etwa $T = 48$ liegt. Dies gilt für die absolute Zahl der Erfolge, aber nicht unbedingt für die Effizienz. Bereits mit einem Agenten sind einige Systeme mit TS komplett erfolgreich, was bei Systemen ohne TS nicht gelingt.

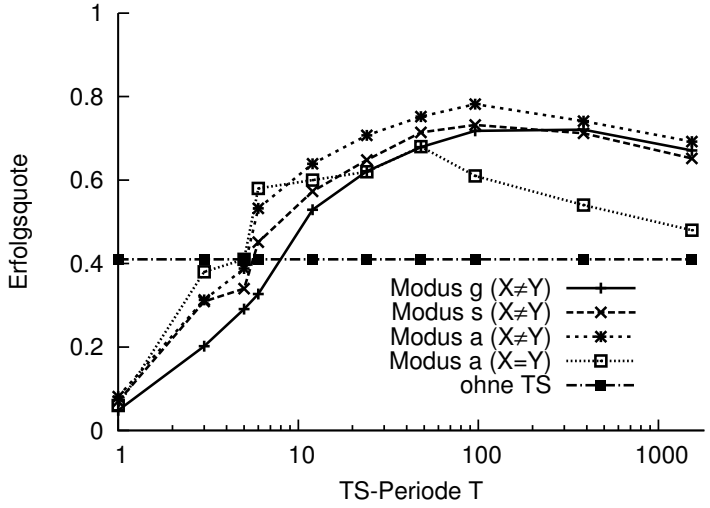

Abb. 6.3.: Mittlere Erfolgsquote der Systeme mit TS in Abhängigkeit der TS-Periode T.

Tab. 6.8 schlüsselt die Kurve des Modus a aus Abb. 6.3 weiter nach der Anzahl der Agenten auf. Mit Erhöhung der Anzahl der Agenten steigt die Anzahl der kompletten Erfolge. Mit $48 \leq T \leq 384$ sind die meisten Erfolge erzielt worden. Es ist eine leichte Tendenz zu erkennen, dass mit mehr Agenten auch höhere TS-Perioden erfolgreicher sind. Die Ergebnisse sind in der Tendenz stellvertretend für die anderen Modi.

Tab. 6.8.: Anzahl der erfolgreichen Systeme (von insgesamt 90) im Modus a mit $X \neq Y$ in Abhängigkeit der Agentenanzahl k und der TS-Periode T.

T	$k=1$	$k=2$	$k=4$	$k=8$	$k=12$	$k=16$	$k=28$	$k=32$	$k=60$	$k=64$	**Summe**
1	0	0	0	5	3	7	11	9	17	21	73
3	0	0	3	17	25	32	41	43	59	62	282
5	0	2	6	23	28	36	52	59	70	73	349
6	1	10	20	33	47	55	69	76	84	84	479
12	7	24	30	48	66	72	81	79	84	84	575
24	8	29	39	70	77	77	84	84	84	84	636
48	**10**	34	**58**	72	77	82	84	86	87	87	677
96	8	**40**	55	**76**	**86**	83	**89**	88	89	**90**	704
384	3	30	48	64	79	**84**	**89**	**90**	**90**	**90**	667
1536	2	12	34	69	74	75	**89**	88	**90**	**90**	623

Die schnellsten und effizientesten Systeme sind in Tab. 6.9 aufgelistet. Das schnellste System ist GG_a-64(96) und benötigt im Mittel 183 Generationen, um eine Welt erfolgreich zu lösen. Alle schnellsten 25 Systeme sind Kombinationen der vier FSMs B, C, G und J. Das effizienteste System ist BJ_a-28(96). In den effizientesten 25 Systemen kommen die FSMs A dreimal und F und E jeweils einmal vor. Der Rest besteht aus B, C, G und J. Das bedeutet, dass Kombinationen von FSMs, die ohne TS effizient sind, besser funktionieren als FSMs, die ohne TS weniger effizient sind. Systeme mit $T = 96$ sind sowohl bei den schnellsten 25 als auch bei den effizientesten 25 Systemen am häufigsten vertreten. Der Modus a ist etwas häufiger als die anderen Modi.

Tab. 6.9.: (a) Schnellste und (b) effizienteste Systeme mit TS und $T \geq 1$.

(a)

k	X	Y	T	**Modus**	G
64	G	G	96	a	183
60	C	J	48	a	192
64	G	J	48	a	194
64	C	J	96	s	196
64	C	J	96	a	196
60	C	J	24	g	197
60	G	J	96	a	198
64	B	C	24	s	200
64	C	J	96	g	202
60	C	G	96	a	206

(b)

k	X	Y	T	**Modus**	C
28	B	J	96	a	10,79
32	B	C	96	a	11,26
28	J	G	96	g	11,30
8	G	J	48	a	11,47
16	J	J	384	a	11,47
28	J	B	384	s	11,50
28	J	G	384	g	11,54
8	J	G	24	s	11,55
8	B	A	384	s	11,58
16	G	J	96	s	11,60

Ergebnisse. Die Ergebnisse zeigen, dass Systeme mit TS-Perioden in einem bestimmten Bereich bessere Ergebnisse erzielen als Systeme ohne TS. Ein „Strategiewechsel“ der Agenten scheint also von Zeit zu Zeit sinnvoll zu sein. Die TS-Technik kann prinzipiell eingesetzt werden, um die Effektivität und die Effizienz der Agenten zu steigern, sie ist also effektiv. Das gilt sowohl für Systeme mit wenigen als auch für Systeme mit vielen Agenten. Die effizientesten Systeme haben dabei 32 oder weniger Agenten. Da die Interaktion der Agenten sich beim CEP auf das Auftauchen von Bewegungskonflikten beschränkt, ist anzunehmen, dass in größeren Welten mehr Agenten effizienter sind. Vermutlich ist eher das Verhältnis der Anzahl der Agenten zur Größe der Welt entscheidend. Diese Frage ist im Zusammenhang mit dem CEP noch nicht systematisch untersucht worden.

6.2 Einsatz von TS und separaten FSMs auf den All-to-All Communication Task

Zwei Experimente unter Verwendung des Beispielsystems ATAC werden in diesem Abschnitt beschrieben. In Abschn. 6.2.1 wird untersucht, welche Art der Evolvierung (gemeinsam oder separat) für Time-Shuffling effektiver und effizienter ist. Dann folgt ein Experiment mit erweiterten Fähigkeiten der Agenten, die Stigmergie und separate FSMs verwenden (Abschn. 6.2.2).

6.2.1 Experiment mit Time-Shuffling und Wrap-Around

Das Experiment in diesem Abschnitt dient dazu, die Frage zu beantworten, wie effektiv der Einsatz der Time-Shuffling-Technik beim ATAC ist und in welcher Art die FSMs dafür gesucht werden sollten. Gleichzeitig soll untersucht werden, wie sich die die Verwendung von Welten mit Rand gegenüber Welten mit Wrap-Around auf das Verhalten und die Leistung der Agenten auswirkt. Zum Teil wurden die Ergebnisse dieses Experiments in [EH09c] und [EH10b] präsentiert.

Der ATAC wird hier in der Grundvariante verwendet. Zwei Sets von Welten sollen von den Agenten gelöst werden. Beide Sets bestehen aus 10 Welten mit je 16 Agenten. Die Platzierung und Ausrichtung der Agenten wurde in jeder initialen Konfiguration zufällig erzeugt. Im ersten Set A (Abb. B.5) haben alle Welten einen Rand und die Größe von 33×33 Zellen (exklusive Rand) ohne Hindernisse. Im zweiten Set B (Abb. B.6) haben alle Welten die Größe 33×33 Zellen und weder Rand noch Hindernisse.

Agenten zum Lösen der Welten sollen mit Hilfe eines GA gefunden werden. Es wird hier ein Inselmodell GA verwendet, mit $I = 7$ Inseln mit je $|P| = 100$ Individuen, einer Elternquote von $q = 0,1$ und einer Migrationswahrscheinlichkeit von $p = 0,02$. Populationen für zukünftige Iterationen werden durch Aussortieren der Individuen mit den schlechtesten Fitnesswerten gebildet. Als Rekombinationstechnik wird UN benutzt, als Mutationstechnik $SG_{0,09}$. Die Fitnessfunktion für eine einzelne Welt j ist

$$f_j = 10^5(16 - c_v) + 10^4(1 - com) + \breve{g}_j$$

wobei c_v die Anzahl der *vollständig informierten* Agenten ist. Ein Agent ist vollständig informiert, wenn er alle Informationen der anderen Agenten bekommen hat. Außerdem gilt $com = 1$, falls mindestens eine Kommunikationssituation aufgetreten ist, und $com = 0$ sonst. $\breve{g}_j$ ist die Anzahl der Generationen, die benötigt wurden, um eine Welt erfolgreich zu lösen. Eine Welt gilt als erfolgreich gelöst, wenn alle Agenten die komplette Information gesammelt haben. Die Funktion realisiert eine Dominanzrelation, die zunächst Systeme, in denen überhaupt eine Kommunikation vorkommt, höher bewertet, und unter diesen solche mit mehr vollständig informierten Agenten besser bewertet. Falls alle Agenten vollständig informiert werden, entscheidet die Anzahl der benötigten Generationen über die bessere Fitness. Die Zehnerpotenzen sind als Faktoren so gewählt, weil die Simulation nach 10.000 Generationen abgebrochen (und als Misserfolg gewertet) wird. Das bedeutet, $\breve{g}$ kann maximal den Wert 10^4 annehmen. Die Gesamtfitness f eines Systems ist der Durchschnitt aller einzelnen Fitnesswerte. Indem für diesen Durchschnittswert statt allen Welten z. B. nur eines der beiden Sets berücksichtigt wird, kann ein Fitnesswert für dieses Set angegeben werden.

Die Inhalte für die Kontrollfunktion sollen mit sechs unterschiedlichen Methoden gefunden werden:

1. Die Kontrollfunktion besteht aus einer einzelnen FSM Z. Sie soll gefunden werden, indem der GA die Fitness jedes Individuums auf allen 20 Welten durch Simulation bestimmt.

2. Die Kontrollfunktion besteht aus zwei FSMs X und Y mit TS-Technik und einer TS-Periode T. Die FSMs werden durch *separates Optimieren* gefunden, FSM X wird auf Set A optimiert, FSM Y auf Set B. Im Anschluss werden die jeweils besten 10 FSMs miteinander kombiniert (200 Möglichkeiten) und mit allen TS-Perioden $1 \leq T \leq 600$ getestet und das beste System bestimmt. Ein auf diese Art evolviertes System wird kurz mit XY_T notiert.

3. Die Kontrollfunktion besteht aus zwei FSMs U und V mit TS-Technik und einer TS-Periode T. Die FSMs werden mit *gemeinsamem Optimieren* und uniformer TS-Rekombination gefunden. Ein auf diese Art evolviertes System wird kurz mit UV_T-u notiert.

4. Die Kontrollfunktion besteht aus zwei FSMs U und V mit TS-Technik und einer TS-Periode T. Die FSMs werden mit *gemeinsamem Optimieren* und TS-Rekombination mit Mittelwertbildung gefunden. Ein auf diese Art evolviertes System wird kurz mit UV_T-m notiert.

5. Die Kontrollfunktion besteht aus zwei FSMs U und V mit TS-Technik und zwei TS-Perioden T_U und T_V. Die FSMs werden mit *gemeinsamem Optimieren* und uniformer TS-Rekombination gefunden. Ein auf diese Art evolviertes System wird kurz mit $U_T V_T$-u notiert.

6. Die Kontrollfunktion besteht aus zwei FSMs U und V mit TS-Technik und zwei TS-Perioden T_U und T_V. Die FSMs werden mit *gemeinsamem Optimieren* und TS-Rekombination mit Mittelwertbildung gefunden. Ein auf diese Art evolviertes System wird kurz mit $U_T V_T$-m notiert.

Die Anzahl der Zustände jeder FSM ist auf $n = 6$ beschränkt. Für die TS-Technik wird immer der Modus a eingesetzt. Beim gemeinsamen Optimieren wird T auf maximal 600 beschränkt.

Von jeder der sechs Methoden wurden 30 unabhängige Durchläufe gemacht. Für die separaten Evolvierungen aus Methode 2 wurde die Anzahl der Iterationen auf 10.000 festgelegt. Die durchschnittliche Zeit, die das Evolvieren von FSMs X auf Set A dauerte, ist $t_{2A} = 3,15$ Stunden, für das Evolvieren von FSMs Y auf Set B wurden im Mittel $t_{2B} = 3,24$ Stunden benötigt. Das Simulieren aller 120.000 Möglichkeiten (200 Kombinationen mal 600 TS-Perioden) auf dem kompletten Set der Welten dauerte im Mittel $t_{2C} = 0,7$ Stunden, so dass die Gesamtdauer für Methode 2 im Durchschnitt $t_2 = 7,09$ Stunden beträgt. Für Methode 1 wurde eine Anzahl von 10.200 Iterationen gewählt, weil die Berechnungszeit dann mit $t_1 = 7,12$ Stunden in etwa der Berechnungszeit von Methode 2 entspricht. In jedem Durchlauf der Methoden 3 bis 6 wurde die Anzahl der Iterationen des GA auf 10.000 festgelegt. Die Dauer der Durchläufe betrug im Mittel $t_3 = 6,33$ Stunden, $t_4 = 6,59$ Stunden, $t_5 = 6,44$ Stunden und $t_6 = 7,37$ Stunden.

Die mit der jeweiligen Methode in den 30 Durchläufen durchschnittlich erreichten Fitnesswerte f_D, sowie die besten und schlechtesten Fitnesswerte sind in Tab. 6.10 eingetragen. Man muss dabei beachten, dass ein niedrigerer Fitnesswert, eine bessere Fitness anzeigt. $f_{\min}$ gibt also den niedrigsten numerischen Wert an, der die Fitness maximiert, $f_{\max}$ den höchsten numerischen Wert. In den Genomteilen, die die FSM definieren, sind aus Platzgründen Leerzeichen und Schrägstriche weggelassen worden.

Das Evolvieren von Systemen ohne TS ist eindeutig die schlechteste Methode. Bei den Systemen mit TS schneiden die Methoden mit gemeinsamem Optimieren deutlich besser ab. Die besten Ergebnisse sind sowohl im Durchschnitt als auch im Maximum bei den Methoden mit

Tab. 6.10.: Beste und durchschnittliche Fitnesswerte aller Methoden nach der gesamten Berechnungszeit.

Typ	t	f_D	$f_{\max}$	$f_{\min}$	bestes System	
Z	7,12 Std.	624,6	654,4	605,6	Z =3L4R3L4R0R2R-3R4L1L5L0R2R	
XY_T	7,09 Std.	566,4	597,8	497,3	X =3R2R4L2R5L4L-3R0L1L5R0R3L Y =2L3L1R4L1R3L-2L0R5R4L1L3R	$T = 377$
UV_T-u	6,33 Std.	432,3	469,3	398	U =2L3L5R5L1R2L-3L2L5R5L0R2L V =0R2R0R2R2L4R-5R4L3R1R5R0L	$T = 18$
UV_T-m	6,59 Std.	415,2	464,4	375	U =5R3R4L2R1L0R-3L4R4L1L1L1R V =5L2R3L2L0L3L-1L2R1L4R1L3L	$T = 40$
U_TV_T-u	6,44 Std.	387,5	462,4	340,1	U =2L3L1L1R0R4L-3L3L1L5R0R4L V =2R2L3L4L0R2L-2R2R0L2R0R4L	$T_U = 2$ $T_V = 30$
U_TV_T-m	7,37 Std.	427,3	455,4	356,9	U =2R5L0L0L2L0R-5L5L1L4L3L1R V =5L3R1L1R0R2R-4L2R1L1R1L3L	$T_U = 60$ $T_V = 36$

zwei TS-Perioden zu finden. Allerdings ist auch die Streuung der Fitnesswerte dort höher. Im Suchraum mit zwei TS-Perioden sind alle Systeme mit nur einer TS-Periode enthalten (alle für die gilt $T_U = T_V$). Die größeren Suchräume birgen aber offenbar auch bessere Systeme in sich, die der verwendete GA in der gewählten Ausführungszeit häufig auch finden kann.

Selbst in den schlechtesten Durchläufen der Methoden mit TS wurde eine Lösung gefunden, die besser ist, als die beste Lösung ohne TS. Gleiches gilt für den Vergleich der separaten und gemeinsamen Optimierung. Die beste gefundene Lösung mit separater Optimierung ist schlechter als alle gefundenen Lösungen der vier Methoden gemeinsamer Optimierung. Bei diesen vier Methoden gibt es allerdings starke Überlappungen, wenn man alle 30 Fitnesswerte der jeweils besten gefundenen Lösung betrachtet (Abb. 6.4, Tab. C.3 bis C.8). Die Systeme U_TV_T-u zeigen in dieser Untersuchung die beste Verteilung.

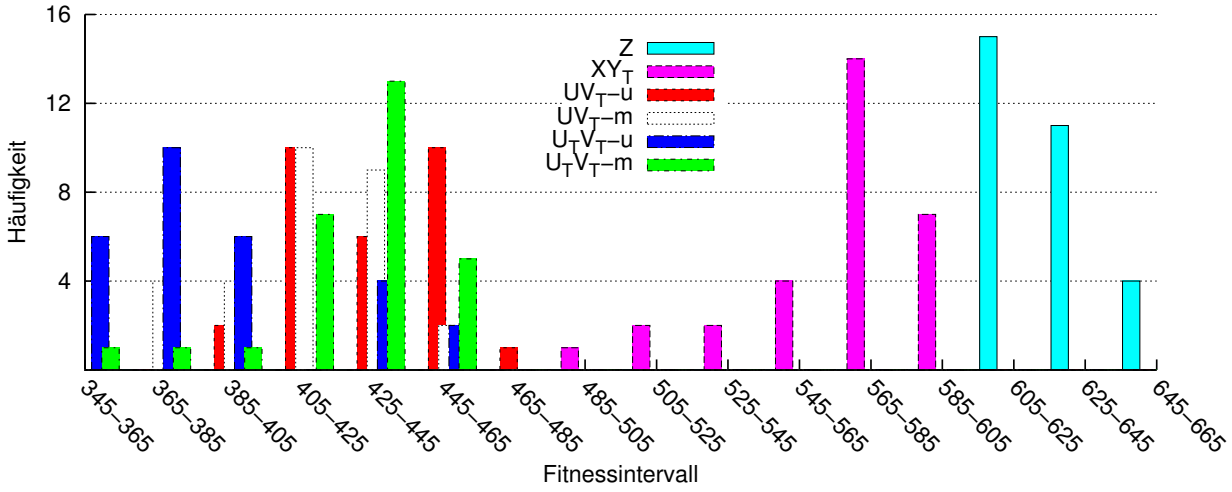

Abb. 6.4.: Verteilung der Fitnesswerte der besten Lösungen aller 30 Durchläufe.

Die Methoden 5 und 6 schneiden am besten ab, wenn man den GA etwa sieben Stunden laufen lässt (Intel Core2Duo mit 2,4GHz und 2 parallelen Threads). Um zu überprüfen, wie die Methoden abschneiden, wenn nur wenig Zeit zum Evolvieren zur Verfügung steht, wurden die Zwischenergebnisse mitprotokolliert. Für die Methode 2 wurden zusätzlich je 30 Durchläufe des

Time-Shuffling gemacht, in denen die FSMs verwendet wurden, die beim separaten Evolvieren nach 2.500, 1.500, 1.000, 400, 200 und 0 Iterationen die beste Fitness hatten. Dadurch reduziert sich die Gesamtberechnung im Mittel auf 2,4, 1,7, 1,4, 1,1, 1,0 und 0,9 Stunden. So können die besten Lösungen jeder Methode zu bestimmten Zeitpunkten verglichen werden (Abb. 6.5).

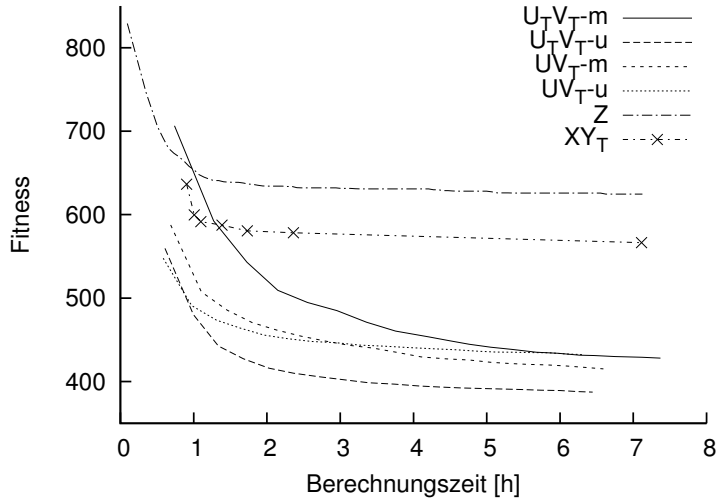

Abb. 6.5.: Die Fitnesswerte der besten Lösungen nach Berechnungszeit, gemittelt über alle 30 Durchläufe.

Es zeigt sich bei der Betrachtung der Kurven deutlich, dass auch nach kürzeren Evolvierungszeiten die erste Methode ohne TS die schlechtesten Lösungen hervorbringt. Separates Optimieren schneidet deutlich schlechter ab als gemeinsames Optimieren. In der Tendenz ist die uniforme TS-Rekombination besser als die Rekombination mit Mittelwertbildung. Auch bei kürzeren GA-Durchläufen ist die Verwendung von zwei TS-Perioden besser. Methode 6 benötigt auffällig viel Zeit für die 10.000 Iterationen, findet aber dennoch nach etwa 1,5 Stunden bessere Lösungen als die Methoden 1 und 2.

Ein weiterer interessanter Aspekt ist die Frage danach, aus welcher Art von FSMs die Systeme mit TS am besten zusammengesetzt werden sollten. Um der Antwort näher zu kommen, wurden die evolvierten FSMs U und V als Kontrollfunktion in den Agenten ohne TS verwendet. Mit den getesteten FSMs konnten weder die Welten des Sets A noch die des Sets B gelöst werden. Die FSMs X und Y wurden ebenfalls als Kontrollfunktion in den Agenten ohne TS verwendet. Mit X (für Set A mit Rand evolviert) konnten die Sets nur teilweise gelöst werden, mit Y (für Set B ohne Rand evolviert) konnten beide Sets gelöst werden, allerdings mit schlechteren Werten als die FSMs Z. Außerdem wurden die FSMs Z in Systemen Z_1Z_{2T} verwendet, indem die besten 10 FSMs eines jeden Durchlaufs mit den 600 TS-Perioden kombiniert wurden. So konnten für jeden Durchlauf Systeme gefunden werden, die eine höhere Fitness als Z aufwiesen, aber niedrigere Werte als die Systeme XY_T.

Insgesamt könnte man sagen, dass man mit den FSMs X und Y zwei „Spezialisten“ evolviert hat, und mit den FSMs Z „Allrounder“. Wendet man auf zwei Allrounder das TS an, kann man die Fitness steigern. Noch besser ist es aber, zwei Spezialisten mit TS zu verwenden. Die beste Lösung für das Gesamtproblem ist allerdings die Verwendung von zwei FSMs, die ausschließlich mit TS funktionieren.

Man kann außerdem beobachten, dass die TS-Perioden in den gemeinsam optimierten Fällen niedriger sind als bei den separat optimierten (Tab. 6.11). Die in der Tabelle eingetragenen durchschnittlichen, minimalen und maximalen TS-Perioden T_D, $T_{\min}$ und $T_{\max}$ für die Methoden 5 und 6 beziehen sich auf die Vereinigungsmenge aller TS-Perioden T_U und T_V. In den Spalten $\varnothing\min(T_U, T_V)$ und $\varnothing\max(T_U, T_V)$ sind die durchschnittlichen Werte der jeweils kleineren TS-Periode bzw. der jeweils größeren TS-Periode eines evolvierten Systems eingetragen.

Es scheint eine gute Strategie zu sein, eine FSM länger aktiv sein zu lassen, wenn sie für eines der beiden Teilprobleme separat evolviert worden ist. Interessant ist, dass bei den Systemen mit zwei TS-Perioden häufig eine kurze und eine lange und nicht etwa zwei gleich lange TS-Perioden evolviert worden sind.

Tab. 6.11.: Durchschnittliche, minimale und maximale TS-Perioden (T_D, $T_{\min}$, $T_{\max}$) der in jedem Durchlauf besten gefundenen Lösung nach der gesamten Berechnungszeit, sowie die Durchschnittswerte der jeweils kleineren bzw. größeren TS-Periode der besten Systeme mit zwei TS-perioden.

Typ	T_D	$T_{\min}$	$T_{\max}$	$\varnothing\min(T_U, T_V)$	$\varnothing\max(T_U, T_V)$
XY_T	244,4	16	418	-	-
UV_T-u	22,2	18	48	-	-
UV_T-m	41,4	38	48	-	-
U_TV_T-u	21,2	1	60	7,9	34,4
U_TV_T-m	47,2	8	88	33,6	60,7

Die in Methode 2 evolvierten Spezialisten X und Y liefern Resultate, die Aufschluss über die Effektivität der Verwendung eines Rands statt eines Wrap-Arounds geben können. Die jeweils besten gefundenen Lösungen haben im Mittel die Fitnesswerte $f = 310,2$ für Set A und $f = 559,3$ für Set B. Die Streuung der Werte ist relativ gering. Es gilt $287,6 \leq f \leq 325,2$ für Set A und $552,1 \leq f \leq 586$ für Set B. Die Welten mit Rand können von den Agenten viel besser gelöst werden. Der Rand ist also viel weniger als Hindernis zu betrachten denn als Orientierungshilfe.

Betrachtet man die Simulationssequenzen, so wird klar, mit welcher Strategie die Agenten arbeiten und dass der Rand eine wichtige Hilfe darstellt (Abb. 6.6). Die Agenten versammeln sich in einer Ecke und bewegen sich dann in den beiden Diagonalen, so dass die Wahrscheinlichkeit für das Auftreten von Kommunikationssituationen erhöht ist. Ohne Rand erzeugen die Agenten mit ihrer Bewegung ein relativ gleichmäßiges Muster, können aber ihre Bewegung nicht auf bestimmte Stellen reduzieren.

Eine typische Strategie der gemeinsam evolvierten Systeme mit TS benutzt ebenfalls den Rand als Hilfe und bewegt sich in der Diagonalen, ist aber deutlich schneller, da durch den Wechsel der FSM nach einer bestimmten Zeit eine Synchronisation stattfinden kann, so dass sich alle Agenten gleichzeitig in die Mitte bewegen. Die großen Pfeile in der Simulationssequenz in Abb. 6.7 geben die grobe Bewegungsrichtung der Agenten an.

Ergebnisse. Die Resultate dieses Experiments zeigen, dass auch beim ATAC Time-Shuffling effektiv sein kann. Die Methode der gemeinsamen Optimierung und die Variante des TS mit zwei TS-Perioden sind am effektivsten und auch am effizientesten, da sie schon nach kurzer Berechnungszeit gute Lösungen liefern. D. h. die Erhöhung der Schwierigkeit bei der Suche wegen des größeren Suchraums wiegt weniger als die Erweiterung der Kapazitäten der Kontrollfunktion nützt. Ob dies auch bei noch größeren FSMs (mit mehr Zuständen) so ist, ist nicht klar.

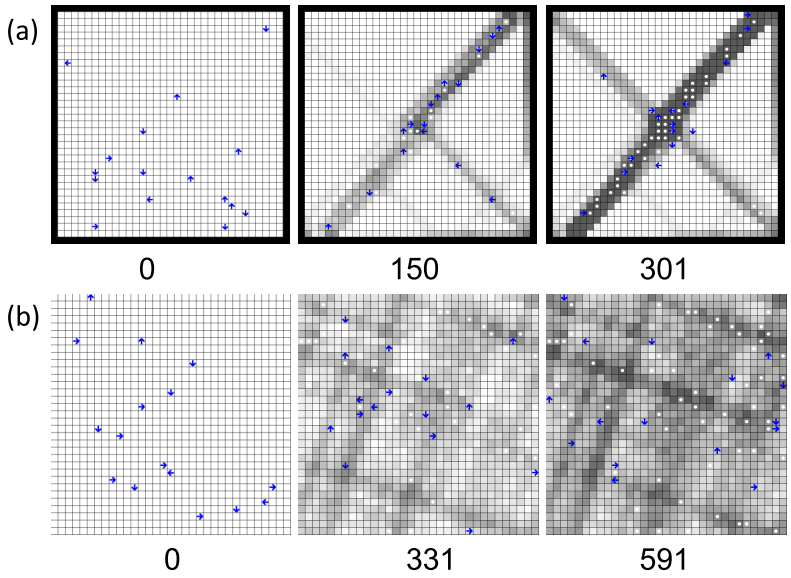

Abb. 6.6.: Beispielsequenzen der speziell evolvierten Agenten im ATAC mit typischen Bewegungsmustern, (a) für Set A, (b) für Set B. Je öfter eine Zelle besucht wurde, desto dunkler ist sie schattiert. Die weißen Punkte geben an, welche Zellen als Mediator fungiert haben.

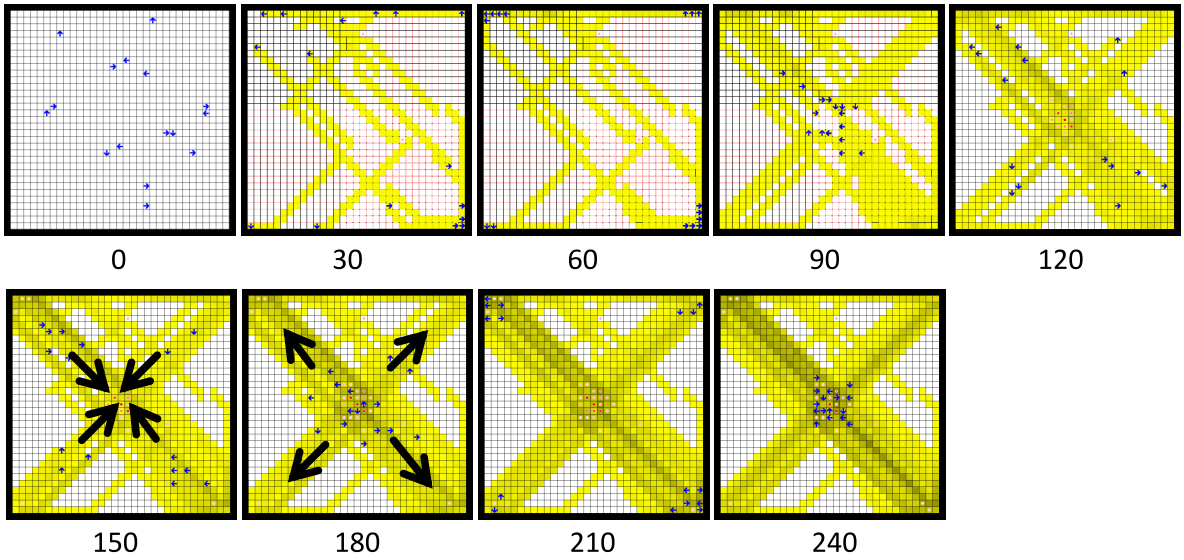

Abb. 6.7.: Beispielsequenz der gemeinsam evolvierten Agenten im ATAC mit typischen Bewegungsmustern und Agenten, die ihre Wege mit Hilfe des Rands synchronisieren.

Die Verwendung von einem Rand kann effektiv sein, weil der Rand für die Agenten eine Hilfestellung darstellt. Einige unsystematische Versuche mit weiteren initialen Konfigurationen, die Hindernisse an verschiedenen Positionen enthielten, wurden gemacht, und dabei festgestellt, dass bei ungeschickter Platzierung der Hindernisse eher ein Nachteil als ein Vorteil für die Agenten besteht.

6.2.2 Experimente mit Stigmergie und separaten FSMs

In den im Folgenden beschriebenen zwei Experimenten geht es darum, herauszufinden, ob eine Erweiterung der Agentenfähigkeiten, die ihnen mehr Bewegungsaktionen und eine indirekte Kommunikation über die Umwelt ermöglichen, einen positiven Effekt auf die Problemlösung hat und ob die durch die Erweiterung komplexer werdenden Agenten effizienter oder weniger effizient evolviert werden können. Außerdem soll untersucht werden, ob die Verwendung von separaten FSMs für die indirekte Kommunikation und die Bewegung effektiv sein kann. Die Ergebnisse zu diesen Untersuchungen sind zum Teil in [EH09d] und [EH10a] publiziert worden.

6.2.2.1 Erstes Experiment mit Stigmergie

Im ersten Experiment wird eine erweiterte Variante des ATAC verwendet. Gegenüber der Grundvariante wird ein neues Objekt eingefügt, das *Flag*. Ein Flag ist ein passives Objekt, das sich auf jeder Zelle befindet. Es hat ein Attribut $F \in \{0, 1\}$. Die Agenten können das Flag auf der Zelle, auf der sie sich gerade befinden, lesen, d. h. es wird als Input für ihre Kontrollfunktion verwendet. Die Idee hinter der Einführung des Flags ist, den Agenten eine zusätzliche Information zu geben, die ihnen bei der Lösung des Problems hilft. Da diese Information von den anderen Agenten kommt bzw. manipuliert wird, kann man von Stigmergie (indirekter Kommunikation) sprechen. Wie genau diese Informationen von den Agenten interpretiert werden oder werden sollen, ist nicht vorgegeben.

Ein weiterer Grund für die Einführung der Flags ist, dass für bestimmte initiale Konfigurationen mit nicht heterogenen Agenten keine Lösung der Welt möglich ist. Das ist dann der Fall, wenn alle Agenten initial die gleiche Richtung besitzen und genug Platz zwischen den Agenten ist, so dass bei einem synchronen Verhalten (alle Agenten haben in jeder Generation die gleichen Inputs und führen die gleichen Aktionen aus) nie eine Kommunikationssituation auftreten kann (Abb. 6.8(a)). Mit Flags kann in vielen Fällen dann trotzdem eine Situation herbeigeführt werden, in der die Agenten nicht mehr synchron reagieren (Abb. 6.8(b)). Allerdings gibt es selbst mit Flags noch Welten, die nicht gelöst werden können. Dann nämlich, wenn die Umwelt für jeden Agenten aus seiner Perspektive trotzdem identisch aussieht, weil die Zwischenräume zwischen den Agenten äquidistant sind (Abb. 6.8(c)). Letztere Situation könnte z. B. durch initial unterschiedlich gesetzte Flags verhindert werden.

Es werden neue Aktionen eingeführt, mit denen die Agenten auch die Möglichkeit haben, das Flag zu verändern. Eine Aktion besteht dabei aus einer Kombination einer *Bewegungsaktion* und einer *Datenaktion* (Veränderung eines Flags). Es gibt drei Datenaktionen:

- $F0$: Das Flag in der Zelle, in der sich der Agent befindet, wird auf $F = 0$ gesetzt.
- $F1$: Das Flag in der Zelle, in der sich der Agent befindet, wird auf $F = 1$ gesetzt.

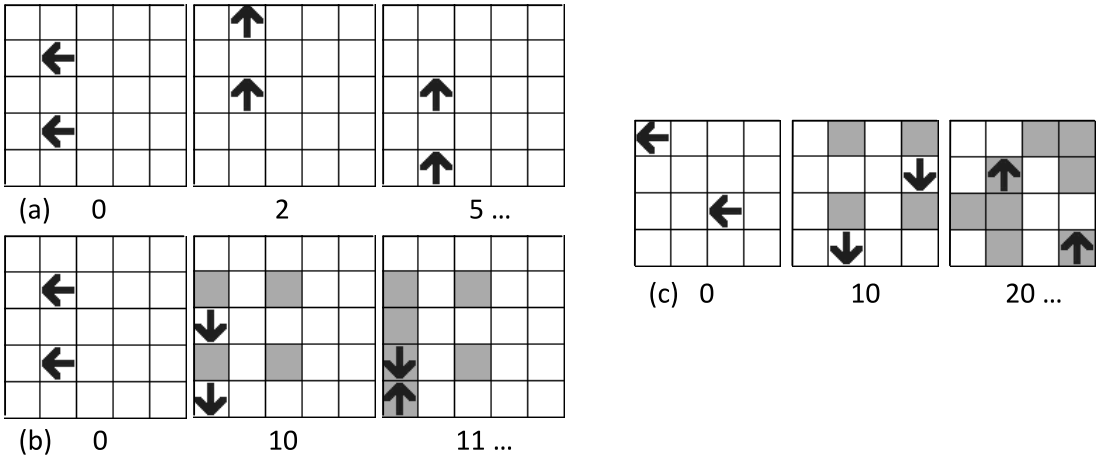

Abb. 6.8.: Beispiele von synchronem Agentenverhalten (a) ohne Flags, (c) mit Flags. (b) Ein Beispiel für die Auflösung der Synchronität durch Flags.

- Fx: Das Flag in der Zelle, in der sich der Agent befindet, behält seinen Wert bei.

Die Menge der Bewegungsaktionen ist gegenüber der Aktionsmenge der Grundvariante ebenfalls erweitert und enthält zusätzlich zu den Aktionen Lm, Rm, Ls, Rs noch Sm und Ss, die so definiert sind, wie beim ARP (Abschn. 4.3). Insgesamt gibt es demnach 18 Aktionen, die der Agent ausführen kann. Diese Veränderungen betreffen auch die Entscheidungsmenge. Die Bewegung auf eine Frontzelle ist wie in der Grundvariante implizit, der Agent entscheidet hier nur zwischen einer Drehung nach links, nach rechts oder keiner Drehung. Zusätzlich entscheidet er zwischen dem Setzen des Flags auf 0, auf 1 oder dem Belassen des Wertes. Insgesamt gibt es also neun Entscheidungsmöglichkeiten: $e \in \{R0, L0, S0, R1, L1, S1, Rx, Lx, Sx\}$, wobei die Endungen $0/1/x$ dafür stehen, dass eine Entscheidung für die Ausführung der Datenaktionen $F0/F1/Fx$ gefallen ist. Konflikte, die der Aktionsarbiter auflösen müsste, gibt es bei den Datenaktionen nicht, da der Agent immer nur das Flag auf der eigenen Zelle verändert.

Durch die erweiterten Inputs und Entscheidungen wird der Suchraum größer ($|K| = (n \cdot 9)^{(n \cdot 4)}$). Zwei Maßnahmen werden getroffen, um den Suchraum einigermaßen klein zu halten bzw. wieder zu reduzieren. Erstens wird die Anzahl der Zustände auf $n = 6$ beschränkt. Zweitens werden verschiedene Varianten von Entscheidungsmengen verwendet, die nur einen Teil der Entscheidungen beinhalten (und damit auch die Aktionsmenge reduzieren). Für die Entscheidungsmengen U und V wird der Input vom Flag nicht benutzt, deshalb ist auch der Exponent im Suchraum kleiner.

Tab. 6.12.: Verwendete Entscheidungsmengen und deren Größe des Suchraums $|K|$ für den ATAC mit Flags.

Typ	U	V	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8
$e \in$	Lx	Lx	$R0$	$R0$	$R0$	$R0$	$R0$	$R1$	$R1$	$R1$	Rx
	Rx	Rx	$L0$	$L1$	$L1$	Lx	Lx	$L1$	Lx	Lx	Lx
		Sx	$S1$	$S0$	$S1$	$S0$	$S1$	$S0$	$S0$	$S0$	$S0$
			Sx	Sx	Sx	$S1$	Sx	Sx	$S1$	Sx	$S1$
$\|K\|$	12^{12}	18^{12}	24^{24}								

Durch einen Vergleich dieser Varianten soll auch ein Rückschluss auf die Effektivität bestimmter zur Verfügung stehenden Aktionen gezogen werden. Es soll neben der Frage nach Effektivität (Kann durch komplexere/andere Entscheidungsmengen die Fitness erhöht werden?) auch die Frage nach der Effizienz (Ist der Optimierungsprozess schneller/langsamer mit komplexeren/anderen Entscheidungsmengen?) beantwortet werden. Die Entscheidungsmengen sind in Tab. 6.12 angegeben. Weitere Kombinationen sind möglich, hier sind aber nur solche ausgewählt worden, die jede der drei Datenaktionen mindestens einmal ermöglichen, die die Möglichkeit, gerade zu laufen enthält und außerdem gleich viele Entscheidungen für Rechts- wie für Linksdrehungen enthalten.

Der ATAC soll von den Agenten auf 20 Welten gelöst werden. Diese entsprechen in Größe und Platzierung der Agenten den Welten aus Abschn. 6.2.1 (Abb. B.5 und B.6). Der initiale Wert aller Flags ist $F = 0$. Das bedeutet, dass keine Information von vorneherein in der initialen Konfiguration ist. Die Agenten müssen sich einen eventuellen Nutzen der Flags sozusagen erst selbst erarbeiten. Die Fitnessfunktion entspricht der aus Abschn. 6.2.1. Der für die Optimierung verwendete GA unterscheidet sich von dem GA aus Abschn. 6.2.1 nur in der Anzahl der Inseln. Hier ist $I = 3$.

Zunächst wurden die Varianten U und V ohne Verwendung von Flags evolviert. Für beide Varianten wurden 30 unabhängige Durchläufe des GA mit je 10.000 Iterationen durchgeführt. Der Fall U ist im Grunde der gleiche, wie die Systeme Z aus Abschn. 6.2.1, allerdings hier mit weniger Inseln im GA. Insofern gibt es hier auch ein anderes Ergebnis. Zum Vergleich sind außerdem *Random Walker* getestet worden. Die Random Walker sind Agenten ohne FSM-Kontrolle, haben aber die gleichen Entscheidungsmengen, aus der in jeder Generation eine Entscheidung zufällig ausgewählt wird. Für beide Varianten sind 1.000.000 Random Walker-Durchläufe simuliert worden.

Die Ergebnisse in Tab. 6.13 zeigen die über alle Durchläufe gemittelte Fitness der besten gefunden Lösung f_D, den Durchschnitt der pro Durchlauf besten zehn gefundenen Lösungen f_{D10} und die gemittelte Fitness der Random Walker $f_{D,RW}$. Im Anhang befindet sich eine Liste der für jede Entscheidungsmenge besten evolvierten FSM (Tab. C.9). Die Random Walker sind keine guten Lösungen im Vergleich zu den evolvierten Agenten. Im Durchschnitt benötigen sie etwa doppelt so viele Generationen, wie die evolvierten Agenten. Der beste Random Walker im Fall V hat einen Fitnesswert von $f = 825$. Die vergrößerte Entscheidungsmenge in V ist effektiv, da die Lösungen deutlich besser sind als für den Fall U. Geradeaus laufen zu können ist für die Agenten offenbar von Vorteil.

Tab. 6.13.: Fitness der evolvierten Agenten und Random Walker ohne Flags.

Typ	f_D	f_{D10}	$f_{D,RW}$
U	640,8	663,2	1.311
V	595,9	615,3	1.038

Die Werte $f_{D10}(W_i)$, also die durchschnittlichen Fitnesswerte der besten zehn gefundenen Lösungen eines Durchlaufs, gemittelt über alle 30 Durchläufe, reichen von 408,4 bis 428,4. Das sind bedeutend bessere Werte als die der evolvierten Agenten ohne die Möglichkeit mit den Flags zu kommunizieren. Trotz der größeren Komplexität der FSMs kann der GA in der gleichen Anzahl von Generationen bessere Lösungen mit den Entscheidungsmengen W_i finden. Abb. 6.9 zeigt die Fitnesswerte $f_{D10}(W_i)$ und $f_D(W_i)$ im Vergleich zu $f_{D10}(V)$ und $f_D(V)$. Die Fälle W_i sind

nach den Werten $f_{D10}(W_i)$ geordnet. Mit der Entscheidungsmenge W_4 konnte die Fitness am deutlichsten gesteigert werden. Der Faktor beträgt $f_D(V)/f_D(W_4) = 1,427$. Außerdem ist die Differenz $f_{D10}(W_i) - f_D(W_i)$ kleiner als $f_{D10}(V) - f_D(V)$. Das spricht dafür, dass der GA mit Flags auch zuverlässiger gute Lösungen finden kann.

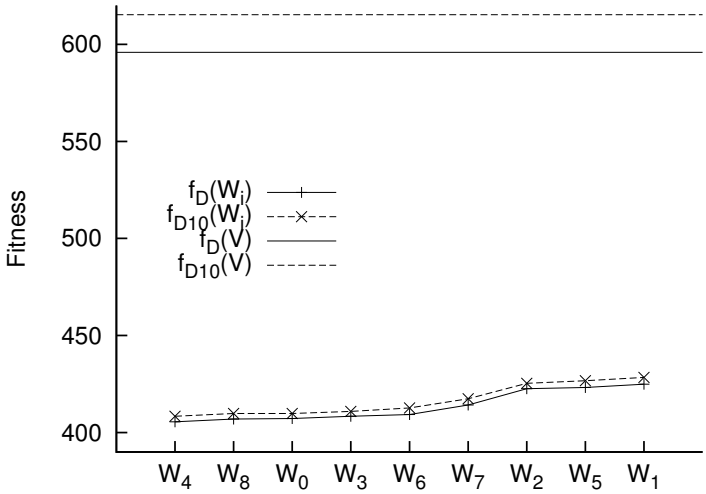

Abb. 6.9.: Fitnesswerte aller Varianten mit Flags.

Die Fälle W_1, W_2, W_5 und W_7 sind unter denen, die Flags verwenden, die schlechtesten. Bei W_1, W_5 und W_7 sind die Entscheidungen Sx und $S0$ enthalten, und in allen vier sind Aktionen enthalten, die den Agenten zwingen, bei einer Drehung ein Flag zu setzen. In den besseren Fällen ist die Aktion $S1$ enthalten. Es scheint, dass die Möglichkeit, beim geradeaus Laufen ein Flag zu setzen, effektiv ist und dass das Setzen des Flags bei einer Drehung kontraproduktiv ist. Um etwas mehr über diesen Zusammenhang herauszufinden, wurden die besten zehn Lösungen aus allen Durchläufen und allen Fällen W_i analysiert. Dabei wurde festgestellt, dass die Agenten bevorzugt geradeaus laufen. In über 70% der Fälle wurde eine der Entscheidungen Sx, $S0$ oder $S1$ getroffen, obwohl diese nur die Hälfte der möglichen Entscheidungen in der Entscheidungsmenge ausmachen und außerdem kein bedeutendes Übergewicht in den Zustandsübergangstabellen haben. Insgesamt scheint es so zu sein, dass das geradeaus Laufen eine gute Strategie für den ATAC ist, und dass die Information, die die Flags bereitstellen, nicht so effektiv kommuniziert werden kann, wenn sie nur bei einem Richtungswechsel ein Flag setzen können oder sogar müssen.

Ein typisches Beispiel für die Art und Weise, wie die evolvierten Agenten die Flags nutzen, ist als Simulationssequenz in Abb. 6.10 dargestellt. Schon nach den ersten Schritten erhöht sich die Anzahl der gesetzten Flags (Zellen mit $F = 1$ sind markiert). Danach bleiben die Flags bis zum Ende der Simulation in etwa konstant gesetzt. Die Agenten bewegen sich dann hauptsächlich in den Bereichen, in denen die Zellen mit einem Flag markiert sind (die besuchten Zellen sind schattiert, je öfter besucht, desto dunkler).

Abb. 6.11 zeigt die unterschiedlichen, typischen erzeugten Bewegungsmuster am Ende einer Simulation. Ohne die Flags erzeugen die Agenten ein (zur Lage des Rands) schräges Muster mit zueinander senkrechten Bewegungslinien. Mit Flags entstehen in den Welten ohne Rand Muster mit senkrechten und waagrechten Linien. Mit Flags sind die Agenten offenbar in der Lage, bevorzugt in die Ecken zu laufen und sich auch dort und in der Nähe des Rands aufzuhalten.

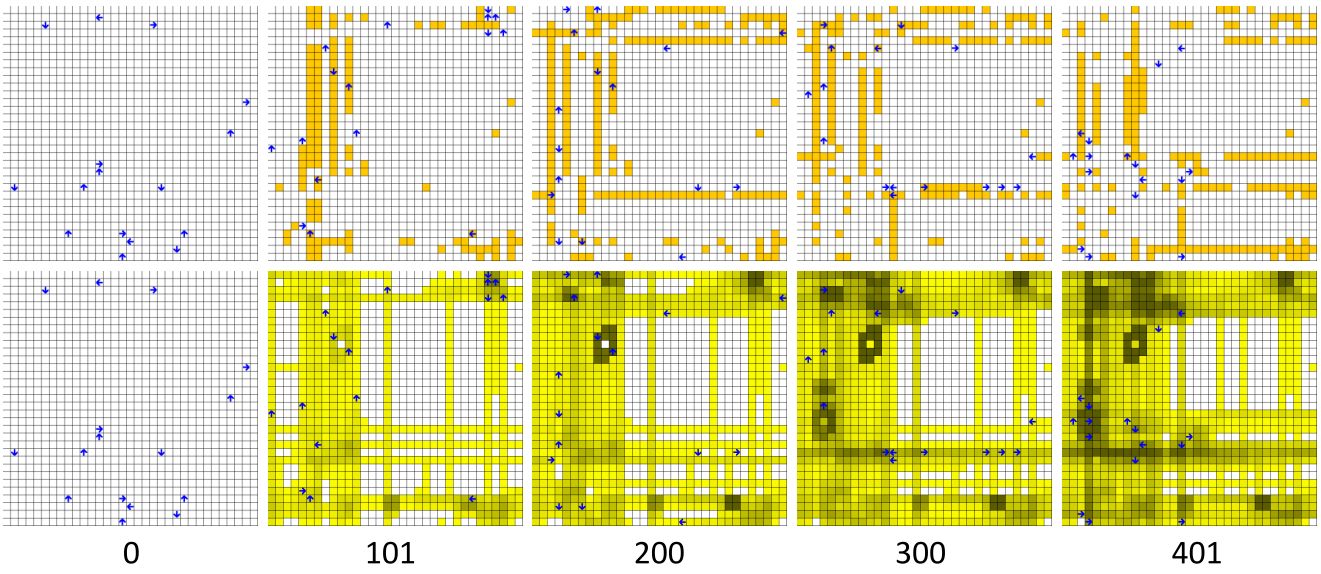

Abb. 6.10.: Muster der Flags (oben) und Bewegungsmuster (unten) in der Simulation der besten gefundenen Lösung des Falls W_4 in einer der Welten. Die Zahlen geben die Generation an.

Sicherlich könnte man dies auch ohne Flags erreichen, jedoch würde ein solches Verhalten bei Welten mit Wrap-Around nicht funktionieren. Die Flags übernehmen hier möglicherweise die Funktion eines Ersatzes für den Rand.

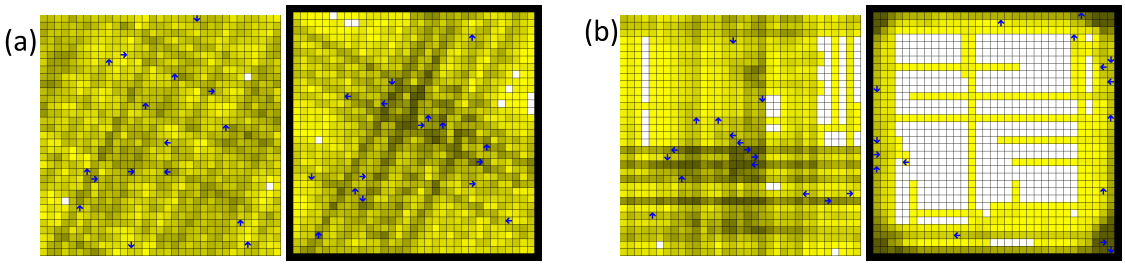

Abb. 6.11.: Erzeugte typische Bewegungsmuster der Agenten mit Flags im ATAC: (a) Fall V, (b) Fall W_4.

Ergebnisse. Bei der Betrachtung der durchschnittlichen Berechnungszeiten (Intel Core2Duo mit 2,4GHz und 2 parallelen Threads) des GA kam heraus, dass im Fall U mehr als 18 Stunden pro Durchlauf gerechnet wurde, im Fall V etwa 15,5 Stunden und in den Fällen W_i etwa 14 Stunden. Das bedeutet, dass die Erweiterung der Aktionsmenge mit Flags nicht nur effektiv, sondern auch effizient ist, da eine bessere Lösung sogar mit weniger Zeitaufwand gefunden wird. Eine Erklärung dafür könnte sein, dass der größte Anteil der Berechnungszeit für die Simulation zur Bestimmung der Fitness aufgewendet wird. Je besser die Fitnesswerte, desto kürzer sind die Simulationen.

6.2.2.2 Zweites Experiment mit Stigmergie

Im zweiten Experiment wurden ebenfalls (binäre) Flags verwendet, aber durch einige Veränderungen die Komplexität der Kontrollfunktion erhöht:

- Die Entscheidungen für die Bewegung und für die Veränderung der Flags werden nun mit zwei separaten FSMs vorgenommen (Abschn. 5.5.2), der *Bewegungs-FSM X* und der *Flag-FSM Y*.
- Außerdem gibt es ein zusätzliches binäres Inputsignal ack für beide FSMs, das ein Rückkopplungssignal für erfolgreiche Kommunikation darstellt. Es liefert eine 1, wenn sich der Kommunikationsvektor in der gegebenen Situation ändert, sonst eine 0. Es ist nicht das gleiche Signal, wie die Bewegungsbedingung m.
- Der Agent kann zwei Flags lesen, das in der eigenen Zelle und das in der Frontzelle. Das Flag in der eigenen Zelle dient als Input für die Flag-FSM Y, das Flag in der Frontzelle dient als Input für die Bewegungs-FSM X. FSM X hat damit insgesamt $i = 8$ mögliche Inputkombinationen, FSM Y hat $i = 4$ mögliche Kombinationen.
- Die Bewegungsaktionen wurden um Bm und Bs mit der Bedeutung wie in Abschn. 4.3 erweitert.
- Die Entscheidungen für die Bewegung auf die Frontzelle ist nicht mehr implizit, sondern wird von FSM X getroffen. Der Aktionsarbiter muss hier allerdings mittels Aktionsmapping Konflikte auflösen und gegebenenfalls eine Bewegung verhindern. Die FSM X kann also acht verschiedene Entscheidungen treffen: $e_X \in \{Lr, Lg, Rr, Rg, Sr, Sg, Br, Bg\}$ mit der Bedeutung, dass die Aktion Xm (move) ausgeführt wird, wenn die entsprechende Entscheidung Xg (go) getroffen wird und keine Konflikte auftreten. Treten Konflikte auf, oder wird eine Entscheidung Xr (rest) getroffen, wird die entsprechende Aktion Xs (stay) ausgeführt.
- Die Flag-FSM Y hat nur zwei mögliche Outputs. Die Entscheidungen bezogen auf die Datenaktionen werden reduziert auf $e_Y \in \{C0, C1\}$, wobei $C0$ das Flag auf $F = 0$ setzt, und $C1$ auf $F = 1$.
- Bei beiden FSMs wird die Anzahl der Zustände auf $n = 8$ beschränkt.
- Die Agenten sollen den ATAC nur auf den zehn Welten mit Wrap-Around lösen.
- Der GA wurde dahingehend modifiziert, dass nun $I = 5$ Populationen verwendet werden. Außerdem wird hier die Mutationstechnik $EV_{0,09}$ benutzt.

Das Rückkopplungs- oder Acknowledge-Signal ack wird eingeführt, weil untersucht werden soll, ob die Agenten aus einer positiven Rückkopplung bei Erfolgen (hier die Verbesserung ihres Kommunikationsvektors) einen Nutzen ziehen können, obwohl es sich um nicht lernende Agenten handelt.

Um den Effekt der Verwendung der Flags und des Acknowledge-Signals bestimmen zu können, werden vier Systemtypen unterschieden (Tab. 6.14). Systeme *A* sind wie eben beschrieben. Systeme *B* sind Systeme ohne das Acknowledge-Signal (und entsprechend reduzierter Komplexität). Systeme *C* haben ein Acknowledge-Signal, aber keine Flags und Systeme *D* verwenden

weder Flags noch ein Acknowledge-Signal. Die letzteren beiden haben somit auch nur eine FSM in der Kontrolleinheit. Für jedes System sind 30 unabhängige Durchläufe des GA gemacht worden.

Tab. 6.14.: Die vier Systemtypen mit separaten FSMs für den ATAC.

Typ	Agentenfähigkeiten		**Größe des Suchraums** $\|K\|$	f_D	f_{D10}	$f_{D10}(D)/f_{D10}$
	Flags	**Inputsignal** ack				
A	•	•	$(64^{64})(16^{32}) \approx 1,34 \cdot 10^{154}$	254,2	263,5	1,64
B	•		$(64^{32})(16^{16}) \approx 1,16 \cdot 10^{77}$	267,9	274,2	1,58
C		•	$(64^{32}) \approx 6,28 \cdot 10^{57}$	446,4	450,6	0,96
D			$(64^{16}) \approx 7,92 \cdot 10^{28}$	426,6	432,9	1

Die Fitnesswerte (Tab. 6.14) sind aufgrund der vielen Veränderungen nicht mit denen aus dem ersten Experiment vergleichbar. Vergleicht man die vier Systeme untereinander, so ist wie auch im ersten Experiment die Verwendung von Flags eine sehr effektive Maßnahme. Trotz des extrem viel größeren Suchraums werden deutlich bessere Lösungen gefunden (bestes System A in Tab. C.10 und C.11). Das Rückkopplungssignal hat, wenn überhaupt, nur einen geringen Effekt. Die durchschnittlichen Fitnesswerte liegen nah beieinander und die Verteilung der Werte ist so ähnlich, dass sie bei nur 30 Durchläufen keinen Aufschluss über einen positiven oder negativen Effekt des Acknowledge-Signals gibt.

In den typischen Bewegungsmustern der besten evolvierten Lösungen (Abb. 6.12) erkennt man, dass die Agenten auf bestimmte gerade Strecken Flags ablegen, und diese Strecken gewissermaßen als Straßen benutzen. Dies ist hier noch deutlicher als im ersten Experiment.

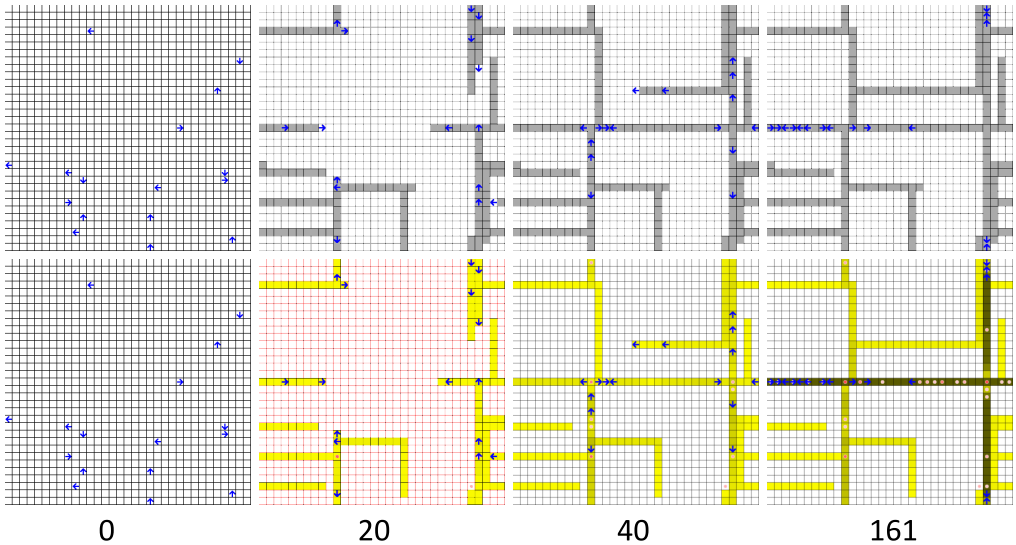

Abb. 6.12.: Muster der Flags (oben) und Bewegungsmuster (unten) in der Simulation der besten evolvierten Lösung. Die Zahlen geben die Generation an.

Die durchschnittlichen Berechnungszeiten eines Durchlaufs (Intel Core2Duo mit 2,4GHz und 2 parallelen Threads) des GA waren im Fall A 17,8 Stunden, im Fall B 14,6 Stunden, im Fall C 13,9 Stunden, und im Fall D 13,4 Stunden. Das bedeutet, dass die Verwendung von zwei FSMs und mehr Inputs effektiv ist, also zu besseren Lösungen führt, aber auch mehr Zeit dafür benötigt. Das bedeutet, dass der GA im Fall A nicht unbedingt effizienter ist als in den anderen

Fällen. Beim Vergleich der besten Fitnesswerte der Populationen über die Zeit fallen keine signifikanten Unterschiede zwischen Fall *A* und *B* auf, der die längere Berechnung erklären könnte. Möglicherweise werden im Fall *A* viele schlechte Kandidaten erzeugt, die lange simuliert werden müssen, um deren Fitness zu bestimmen, die aber nie in die Population aufgenommen werden und deshalb nicht mitprotokolliert wurden.

Reduziert man im Fall *A* die Berechnungszeit auf etwa 13,4 Stunden im Durchschnitt, so entspricht das in etwa einer Anzahl von 8.000 Iterationen des GA. Die zu diesem Zeitpunkt erreichte durchschnittliche Fitness der besten zehn Lösungen ist $f_{D10}(A) = 278,3$. Das ist immer noch eine deutliche Steigerung zur Fitness im Fall *D*. Für Fall *B* ergeben sich 9.000 Iterationen und eine Fitness von $f_{D10}(B) = 280,5$, für Fall *C* 9.500 Iterationen und $f_{D10}(C) = 451,9$.

Ergebnisse. Die Einführung von zusätzlichen Bewegungsaktionen und Flags zur Ermöglichung von Stigmergie, sowie die Verwendung von separaten FSMs können effektiv sein. In den beiden Experimenten konnte die Leistung der Agenten bzw. die Fitness der vom GA gefundenen Lösungen im Durchschnitt um bis zu 42% bzw. 64% gesteigert werden. Allerdings führte die Erweiterung zumindest im zweiten Experiment auch zu einer höheren Berechnungszeit. Bei gleicher Berechnungszeit ist noch eine Steigerung um 53% zu verzeichnen. Das Acknowledge-Signal für die Agenten brachte keinen wahrnehmbaren Effekt.

6.3 Anpassung der Weltgröße und Einsatz von randomisierten FSMs auf das Agent Routing Problem

In Abschn. 6.3.1 wird ein Experiment beschrieben, das Aufschlüsse über die optimale Größe der Agentenwelt und die Dichte der Agenten darin geben soll, danach wird in Abschn. 6.3.2 untersucht, wie randomisierte FSMs einsetzbar sind.

6.3.1 Experiment mit der Weltgröße und der Anzahl der Agenten

Für das Design eines Routers (z. B. auf einem Hardware-Chip oder auch als Modell für andere Transportsysteme) müssen viele Parameter berücksichtigt werden. Die Anzahl der Knoten, die Verbindungstopologie und Größe des Netzes, der Routing-Algorithmus etc. In diesem Abschnitt wird eine Untersuchung beschrieben, die darauf abzielt, die optimale Struktur eines Routers zu finden. Dazu wird die Größe der Welt (des Routers) und das Verhältnis der Größe zur Anzahl der Agenten variiert, um den Einfluss dieser Variablen auf den Durchsatz von Nachrichten zu bestimmen. Der Routing-Algorithmus wird in Form von FSM-kontrollierten Agenten evolviert. Die Resultate dieser Untersuchung wurden teilweise in [EH10d] veröffentlicht.

Es wird im Wesentlichen die Grundvariante des ARP verwendet, allerdings mit einer anders gestalteten Inputreduktionstabelle, so dass statt sechs sieben Zonen, in denen sich die Zielposition relativ zur Agentenposition und Agentenrichtung befinden kann, unterschieden werden (Abb. 6.13).

In diesem Experiment werden Welten mit Rand verwendet. Die Zielpositionen der Agenten sind gleichzeitig Startpositionen von anderen Agenten, wobei nie zwei Agenten dieselbe Zielposition haben. Zwölf verschiedene Fälle mit unterschiedlichen Weltgrößen und unterschiedlicher Anzahl von Agenten werden betrachtet (Abb. 6.14):

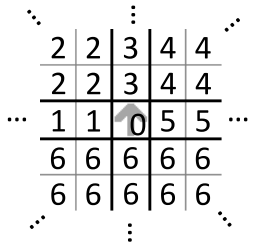

Abb. 6.13.: Alternative sieben Zonen der Zielpositionen im ARP.

- Acht Fälle A_1-A_8: Dies sind Router (Welten) mit vier Agenten. In den Fällen A_1 und A_2 gibt es keine freien Zellen. Die Bewegungen erfolgen zumindest am Anfang nur durch Swaps. In den anderen Fällen gibt es je zwei freie Zellen, die von den Agenten benutzt werden können, um besser Konflikte aufzulösen.

- Vier Fälle $B(k)$, $k = 16, 64, 144, 256$: Im diesem Fall gibt es fünf verschiedene Typen von initialen Konfigurationen der Welten mit je k Agenten. $B(k)$-0 bedeutet, dass die Agenten in einer initialen Konfiguration der Größe $\sqrt{k} \times \sqrt{k}$ ohne freie Zellen platziert werden. $B(k)$-1 bedeutet, dass die Agenten in gleicher Weise angeordnet werden, aber eine Zeile bzw. Spalte von Zellen zwischen den Agenten und dem Rand eingefügt wird. $B(k)$-1R ist eine Welt der gleichen Größe wie $B(k)$-1, nur dass hier die Agenten und damit auch die Zielpositionen zufällig platziert werden. $B(k)$-2 und $B(k)$-2R werden auf gleiche Weise konstruiert, nur mit zwei zusätzlichen Zeilen (Spalten) von Zellen statt einer.

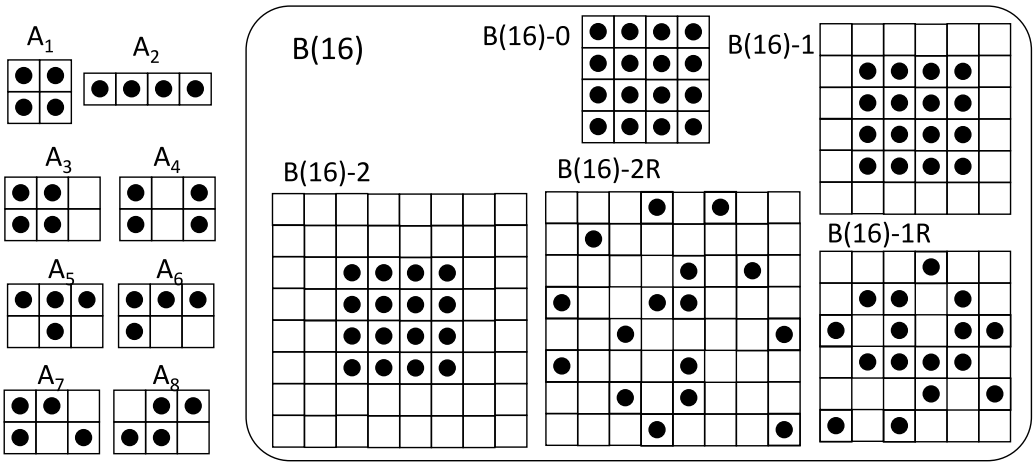

Abb. 6.14.: Die untersuchten Fälle zur Bestimmung der optimalen Struktur eines Routers.

FSMs wurden für jeden der zwölf Fälle separat optimiert. Dazu wurde ein Inselmodell-GA eingesetzt. Die Parameter sind $I = 5$ Inseln, Populationsgröße $|P| = 100$, Migrationswahrscheinlichkeit $p = 0,02$. Die verwendete Technik für Rekombination und Mutation ist $UN\text{-}SG_{0,09}$.

In den Fällen A_i wurden 80 verschiedene initiale Konfigurationen (Richtung der Agenten und Permutation der Zielpositionen) als *Training Set* zum Optimieren verwendet. Für die Fälle $B(k)$ sind je 20 initiale Konfigurationen (vier von jedem der fünf Typen) zum Optimieren verwendet worden. Für jeden Fall sind 6 unabhängige Durchläufe des GA mit 10.000 Iterationen gemacht worden. Die Anzahl der Zustände ist dabei auf $n = 6$ begrenzt worden.

Die Fitnessfunktion ist eine Dominanzrelation, bei der folgende Größen in dieser Reihenfolge berücksichtigt werden:

1. Die Anzahl der Welten, in der die Agenten alle Ziele innerhalb von 2.500 Generationen erreichen, d. h. die Anzahl der erfolgreich gelösten Welten E.
2. Die Summe $\sum_{j=1}^{J} \bar{t}_j$ der nicht erreichten Zielpositionen nach 2.500 Generationen, mit $J = 20$ in den B-Fällen bzw. $J = 80$ in den A-Fällen. $\bar{t}_j$ ist dabei die Anzahl der nicht erreichten Zielpositionen (targets) in einer Welt j.
3. Die mittlere Dauer G wie in Abschn. 6.1.1, die nur für $E = J$ definiert ist.

Für komplett erfolgreiche Lösungen (die in jeder Welt alle Zielpositionen erreichen) ist die Fitness äquivalent zur Anzahl der Generationen, die sie dafür benötigen. Die Gesamtfitness ist der Durchschnitt der Fitnesswerte aller Welten.

Aus allen sechs Durchläufen wurden die besten gefundenen Lösungen zu einer Liste mit 100 FSMs zusammengefasst. Diese 100 FSMs wurden auf einem erweiterten *Ranking Set* getestet, um deren Robustheit gegenüber Änderungen der Welt zu bewerten. In den A-Fällen bestand das Ranking Set aus allen 2304 Permutationsmöglichkeiten. In den B-Fällen wurden 400 zufällige initiale Konfigurationen generiert (80 von jedem Typ).

Im Fall A_1 konnte keine komplett erfolgreiche FSM gefunden werden, weil sich ein unauflösbarer Deadlock konstruieren lässt (s. auch Abschn. 5.5.5). Für den Fall A_2, der ebenfalls ohne freie Zellen auskommt wurde eine FSM gefunden, die alle 2304 Welten in durchschnittlich $G = f_{A2} = 3,89$ Generationen lösen kann. Es ist also nicht in allen Fällen notwendig, einige Zellen leer zu lassen. Von den anderen Fällen wurden nur in A_6 und A_8 komplett erfolgreiche Lösungen gefunden, mit einer Fitness von $f_{A6} = 4,7$ bzw. $f_{A8} = 4,4$. Durch die freien Zellen kann die Leistung der Agenten nicht verbessert werden. Die etwas längeren Wege führen sogar zu einer schlechteren Fitness.

In allen B-Fällen wurden mindestens 100 FSMs gefunden, die das Training Set komplett lösen konnten. Das erweiterte Ranking Set wurde jedoch nur von FSMs aus Fall $B(256)$ komplett gelöst (Tab. 6.15). Eine gute Fitness auf dem Training Set ist in diesem Fall kein Hinweis auf die Robustheit der FSM. Es müsste entweder das Training Set oder die Welt vergrößert werden, um robustere FSMs zu finden. Die maximale Erfolgsquote wurde auch für die Ranking Sets der jeweils anderen drei Fälle bestimmt. Im Fall $B(256)$ wurden 28, 55, 63 bzw. 61 FSMs gefunden, die die einzelnen Ranking Sets komplett erfolgreich lösten. Vier der 100 FSMs konnten alle vier Ranking Sets erfolgreich lösen (Tab. C.12). Die Werte sind insgesamt ein Hinweis darauf, dass für schwierigere Probleme evolvierte FSMs auch auf leichteren Problemen funktionieren, aber nicht umgekehrt.

Tab. 6.15.: Maximale Erfolgsquote auf den Ranking Sets. Die Werte in Klammern geben die Anzahl der Lösungen an, die komplett erfolgreich waren.

evolviert für	**Ranking Set** ($J = 400$)			
	$B(16)$	$B(64)$	$B(144)$	$B(256)$
$B(16)$	365	152	19	1
$B(64)$	400 (1/100)	393	356	293
$B(144)$	400 (5/100)	394	377	325
$B(256)$	400 (61/100)	400 (63/100)	400 (55/100)	400 (28/100)

Die theoretisch maximal erreichbare Fitness in einer Welt ist mindestens durch das Maximum aller Manhattan-Distanzen (kürzeste Wege) zwischen Startpositionen und Zielpositionen der Agenten beschränkt. Dieser Wert kann unter Umständen nicht erreicht werden, wenn Konflikte nicht vermieden werden können und weil die initiale Richtung der Agenten einen oder zwei zusätzliche Schritte notwendig machen kann. Dieses Optimum beträgt im Mittel für das Ranking Set im Fall $B(256)$ 28,28. Die beste evolvierte Lösung hat eine Fitness von $f_{B(256)} = 87,6$, also mehr als das dreifache des Optimums. Eine generelle Beobachtung ist, dass die nicht ganz so erfolgreichen FSMs schneller sind als die robusteren, natürlich nur auf die erfolgreich gelösten Welten bezogen.

Weitere Tests mit Sets von initialen Konfigurationen $B(256)$-zR mit $z \in \{1, 2, \ldots 10\}$ wurden durchgeführt, um das beste Verhältnis von Agentenzahl zu Feldgröße bzw. Anzahl an freien Zellen zu bestimmen. Die Liste der besten 100 evolvierten FSMs wurde auf 80 zufällig erstellten Welten für jedes z simuliert. Am schnellsten (mit Fitness $f = 78,76$) konnten die FSMs die Welten mit $z = 5$ lösen (Abb. 6.15). In diesen Welten ist das Verhältnis von leeren Zellen zu von Agenten besetzten Zellen initial bei $420/256 = 1,64$. Je größer der Anteil an freien Zellen ist, desto weniger Konflikte treten auf, daher nähert sich die Kurve der Fitness mit steigender Größe der Welt dem Optimum an. Die Werte suggerieren, dass es ein optimales Verhältnis im Bereich von 1 bis 2 gibt, da die Kurve (eigentlich deren Interpolation) einen monotonen Verlauf ohne weitere lokale Optima hat. Allerdings sind die FSMs auch nicht für die Spezialfälle mit wenig Konflikten evolviert worden.

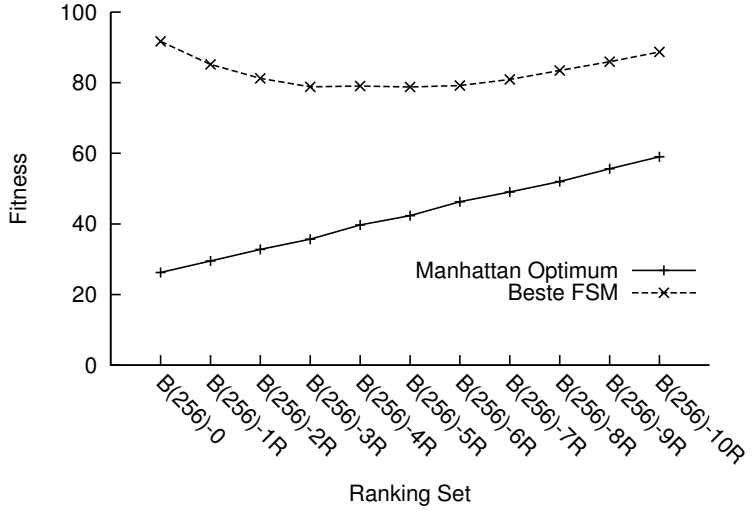

Abb. 6.15.: Die Fitnesswerte der besten FSM aus der Liste der besten 100, abhängig von der Anzahl der freien Zellen.

Eine Reihe weiterer Simulationen auf speziellen Welten wurde durchgeführt, um die evolvierte Strategie der Agenten zu analysieren und festzustellen, ob eine regelmäßige Anordnung der Start- und Zielpositionen einen Effekt hat. Es wurden dafür schachbrettmusterartige Welten der Größe 24×32 und 16×32 entworfen (Abb. 6.16). Das Verhältnis von freien Zellen zu Agenten beträgt dann exakt 1 bzw. exakt 2. Je 400 verschiedene Permutationen der Zuordnung der Zielpositionen wurden verwendet. Die mittlere erreichte Fitness ist $f = 82,4$ bzw. $f = 79,9$. Die Werte sind leicht, aber nicht signifikant schlechter als die bei den unregelmäßigen Anordnungen, so dass über die Struktur des Routers bezogen auf die Anordnung der Agenten keine Schlussfolgerung gezogen werden kann.

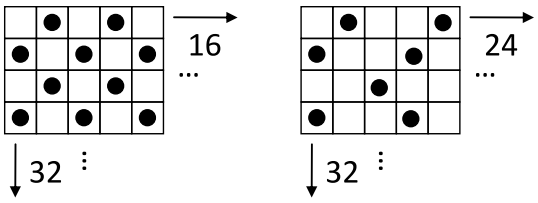

Abb. 6.16.: Regelmäßige Anordnungen im ARP.

In Abb. 6.17 ist eine Beispielsequenz der für $B(256)$ besten evolvierten FSM dargestellt. Auf diesem per Hand konfigurierten Zellfeld sollen die Agenten über Kreuz ihr Ziel auf der um den Mittelpunkt symmetrisch gespiegelten Zelle erreichen, d. h. der Agent an Position (x, y) findet sein Ziel an der Position $(23 - x, 31 - y)$ bei einer Weltgröße von 24×32 Zellen. Die FSM benötigt hier 158 Generationen, also etwa doppelt so viele, wie bei den zufälligen Anordnungen. Zunächst formiert sich ein Cluster in der Mitte der Welt. Danach laufen die Agenten im Uhrzeigersinn um den Cluster herum zu ihrem Ziel.

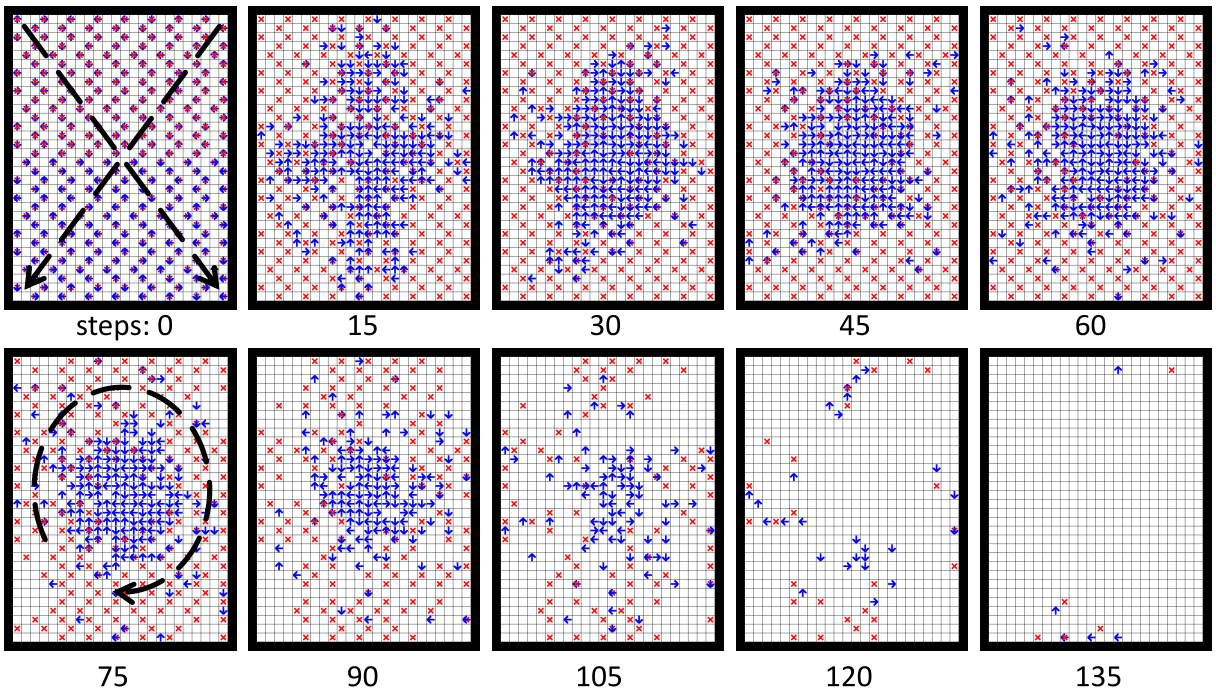

Abb. 6.17.: Beispielsequenz im ARP auf einer regelmäßigen Routerstruktur.

Ergebnisse. In Welten mit weniger freien Zellen ist es wahrscheinlicher, dass die Agenten in einen Livelock geraten. Außerdem bieten einige freie Zellen den Agenten mehr Platz, um Konflikte zu vermeiden. Allerdings vergrößern sich auch die Distanzen zwischen Start- und Zielposition, wenn mehr freie Zellen eingefügt werden. Für die in diesem Experiment besten evolvierten FSMs ist ein Verhältnis von Agenten zu freien Zellen zwischen 1 und 2 optimal. Die FSMs, die auf schwierigen Problemsets evolviert worden sind, waren robust genug, um auch die einfachen Problemsets zu bewältigen, allerdings langsamer als die für die leichteren Problemsets evolvierten FSMs.

6.3.2 Experiment mit randomisierten FSMs

Randomisierte FSMs sollen im ARP speziell dafür eingesetzt werden, um Deadlock-Situationen aufzulösen. In diesem Abschnitt wird ein Experiment beschrieben, mit dem untersucht werden soll, inwieweit randomisierte FSMs dafür eingesetzt werden können, ob es durch den Zufall zu großen Performanzverlusten kommt und mit welcher Methode der Zufall in den Agenten eingebaut werden soll. Teile dieser Untersuchung wurden in [EH10c] publiziert.

In diesem Experiment wird eine Variante des ARP verwendet, die sich von der Grundvariante nur in der Inputreduktionstabelle unterscheidet. Der Agent kann neun Zonen unterscheiden, in denen sich seine Zielposition befinden kann (Abb. 6.18). Die Lage der Zielposition wird relativ zur Agentenrichtung erfasst und außerdem, im Unterschied zur Grundvariante, ausgehend von der Frontzelle. Dies erscheint deshalb eine sinnvolle Veränderung, weil der Agent, wenn kein Konflikt besteht, gezwungen ist, sich auf die Frontzelle zu bewegen und dann dort eine neue Richtung annehmen kann, um sich in der Folge besser in Richtung der Zielposition bewegen zu können.

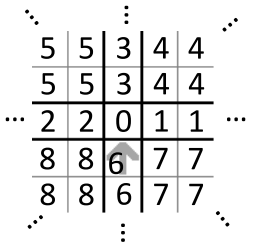

Abb. 6.18.: Zielsegmente im ARP für randomisierte Agenten.

Konflikte im Allgemeinen und Deadlock-Situationen treten häufiger auf, wenn das Zellfeld dichter mit Agenten besetzt ist. Da hier der Effekt der Randomisierung im Besonderen auf die Deadlocks untersucht werden soll, werden in diesem Experiment nur Welten verwendet, die in ihrer initialen Konfiguration in jeder Zelle einen Agenten haben. Die Zielpositionen der Agenten sind gegenseitig exklusiv verteilt, d. h. es gibt keine zwei Agenten mit der gleichen Zielposition, was zur Folge hat, dass jede Zelle für einen Agenten eine Zielposition sein muss. Die Start- und Zielposition eines Agenten sind verschieden. Drei Fälle von verschieden großen initialen Konfigurationen werden unterschieden. Alle Welten haben quadratische Ausmaße $N \times N$, wobei $N^2 = k$ mit $k \in \{16, 64, 256\}$.

Für jeden der drei Fälle wurden 20 initiale Konfigurationen mit zufälliger Zuordnung von Start- und Zielpositionen und zufälliger Bewegungsrichtung erstellt. Diese *Training Sets* werden verwendet, um in einem Inselmodell-GA entsprechend Abschn. 5.4.3 randomisierte FSMs zu optimieren. Die gewählten Parameter sind: Populationsgröße $|P| = 100$, Anzahl der Inseln $I = 5$, Migrationswahrscheinlichkeit $p = 0,02$, Elternquote $q = 0,1$. Die verwendete Technik für Rekombination und Mutation ist $UN\text{-}SG_{0,09}$. Die Zufallswahrscheinlichkeit p für die Auswahl einer zufälligen Entscheidung wird mit Mittelwertbildung rekombiniert.

Die Zufallswahrscheinlichkeit kann im kompletten Intervall $0 \leq p \leq 1$ liegen, ist aber auf drei Nachkommastellen Genauigkeit beschränkt, so dass es 1.000 mögliche Werte gibt. Die Anzahl der Zustände der FSM ist auf $n = 8$ beschränkt worden. Die Gesamtanzahl der möglichen Inputkombinationen beträgt $i = 18$ (neun Zonen und zwei Bewegungsbedingungen). Mit vier Entscheidungsmöglichkeiten ergibt sich ein Suchraum von $1000 \cdot 32^{144} \approx 5,5 \cdot 10^{219}$.

Tab. 6.16.: Beste Fitnesswerte und Zufallswahrscheinlichkeiten der randomisierten FSMs im ARP.

	$k = 16$		$k = 64$		$k = 256$	
	f	p	f	p	f	p
beste FSM nach Anpassung von p	14,8	1,2%	36,7	1,3%	84,0	1,8%
beste FSM vor Anpassung von p	15,4	4,5%	37,6	0,6%	86,4	4,4%
$mXY(p)$	15,8	6,5%	41,1	9,1%	97,8	11,9%
$XY(p)$	17,9	8,4%	50,2	14,7%	122,7	18,5%

In jedem Fall wurden sechs unabhängige Durchläufe des GA mit jeweils 10.000 Iterationen gemacht. Die Fitnessbewertung ist die gleiche Dominanzrelation wie in Abschn. 6.3.1. Trotz des nicht deterministischen Verlaufs einer Simulation wurde aus Zeitgründen zur Fitnessbestimmung nur jeweils eine Simulation durchgeführt. Am Ende der GA-Durchläufe wurde dann für alle 3000 FSMs (6 GAs mit je 5 Inseln mit je 100 FSMs) eine Fitnesskorrektur durchgeführt. Dafür wurde jede FSM 1000 mal auf dem Training Set simuliert und der im Mittel erzielte Fitnesswert als eigentliche Fitness verwendet, um die Lösungskandidaten nach Qualität zu sortieren.

Aus den sortierten Listen der evolvierten FSMs wurden die besten 20 randomisierten FSMs auf einem größeren *Ranking Set* von 1000 Welten (für jeden Fall k separat) je 10.000 mal (1000 mal für $k = 256$) simuliert. Zwei von diesen Welten sind manuell erstellt worden und enthalten für nicht randomisierte Agenten unauflösbare Deadlocks. Sie entsprechen dem Schema aus Abb. 5.16(b). Die Zufallswahrscheinlichkeit der FSMs wurde bis dahin nur für das Training Set bestimmt. Um für jede der 20 FSMs einen optimalen Wert für das Ranking Set zu bekommen, wurden die Simulationen mit verschiedenen Werten durchgeführt und in iterativen Schritten mit 0,1% Genauigkeit angepasst.

Zum Vergleich mit den randomisierten FSMs wurden zwei weitere Kontrollfunktionen mit gleichen Inputs und Outputs verwendet. Die erste ist ein randomisiertes XY-Routing. Agenten, die so kontrolliert sind, bewegen sich zuerst in X-Richtung auf Höhe ihrer Zielposition, danach in Y-Richtung auf ihr Ziel zu. Die Agenten haben ebenfalls eine Zufallswahrscheinlichkeit p, mit der sie eine zufällige Entscheidung treffen. Diese Agenten werden als $XY(p)$ bezeichnet.

Die zweite Kontrollfunktion ist ein modifiziertes XY-Routing ($mXY(p)$), bei dem der Agent in jedem Schritt zufällig in X-Richtung oder in Y-Richtung die Distanz zur Zielposition verringert. Ist nur eines von beiden möglich, weil er sich in einer der beiden Richtungen schon auf Höhe der Zielposition befindet, dann kann er sich auch nur in diese eine Richtung bewegen. Auch die modifizierte XY-Routing Kontrollfunktion besitzt eine Zufallswahrscheinlichkeit. Diese Kontrollfunktionen würden in MAS ohne Konflikte und ohne zufällige Entscheidungen immer den kürzesten Pfad laufen. Für das Ranking Set wurde der optimale Wert der Zufallswahrscheinlichkeit genau wie bei den randomisierten FSMs bestimmt.

Die Fitness der besten FSMs (Tab. C.13) nach bzw. vor der Anpassung der Zufallswahrscheinlichkeit und die Fitnesswerte der XY-Routing-Agenten und der modifizierten XY-Routing-Agenten sind in Tab. 6.16 gegeben. Die evolvierten FSMs sind in allen drei Fällen schneller als die anderen Kontrollfunktionen. Je größer die Welt, desto größer wird auch der prozentuale Abstand der Fitnesswerte. Durch die Anpassung der Zufallswahrscheinlichkeit konnte die Fitness der evolvierten FSMs nochmal um zwei bis vier Prozent gesteigert werden.

Eine weitere Beobachtung ist, dass die FSMs deutlich kleinere Zufallswahrscheinlichkeiten haben als die beiden alternativen Kontrollfunktionen. Unter den FSMs ist das Phänomen zu

Tab. 6.17.: Durchschnittliche, minimale und maximale Zufallswahrscheinlichkeiten im ARP mit randomisierten Agenten.

	nach Anpassung von p			**vor Anpassung von p**		
	$k = 16$	$k = 64$	$k = 256$	$k = 16$	$k = 64$	$k = 256$
Durchschnitt	3,1%	1,89%	2,19%	3,1%	4,78%	5,69%
Minimum	0,9%	0,8%	1,2%	0,1%	0,6%	2,9%
Maximum	5,6%	3,1%	3,2%	6,1%	7,6%	7,5%

beobachten, dass eine bessere Fitness meist mit einer niedrigeren Zufälligkeit einhergeht. In Tab. 6.17 sind die minimalen, maximalen und durchschnittlichen Zufallswahrscheinlichkeiten der 20 besten evolvierten FSMs (nach Ranking Set-Evaluation) vor und nach Anpassung von p angegeben. Für die nicht angepassten Werte, sowie für die XY- und mXY-Routing-Agenten gilt, dass mit steigendem k auch p steigt. Nach der Optimierung der Zufallswahrscheinlichkeiten gilt dieser Zusammenhang allerdings nicht mehr. In der Tendenz bedeutet es, dass für eine höhere Anzahl von Agenten und potentiell mehr Konfliktsituationen eine höhere Anzahl an zufälligen Aktionen notwendig ist, um diese auflösen zu können.

Die Optimierung der Zufallswahrscheinlichkeiten bedeutete in allen Fällen immer eine Verringerung. Auch das Verteilungsintervall der Werte für p auf alle 20 FSMs bezogen ist kleiner geworden. Die Werte sind möglicherweise deshalb zu hoch, weil bei der Evolvierung nur eine einzige Simulation stattfand, um die Fitness zu evaluieren. Bei hohen Zufallswahrscheinlichkeiten ist die durchschnittliche Leistung nicht so gut, weil jede sinnvolle Strategie dadurch verloren geht, aber hin und wieder landet man einen „Glückstreffer“. Dieser ist dann bei dem Verfahren dieses GA nicht mehr so einfach aus der Population zu verdrängen.

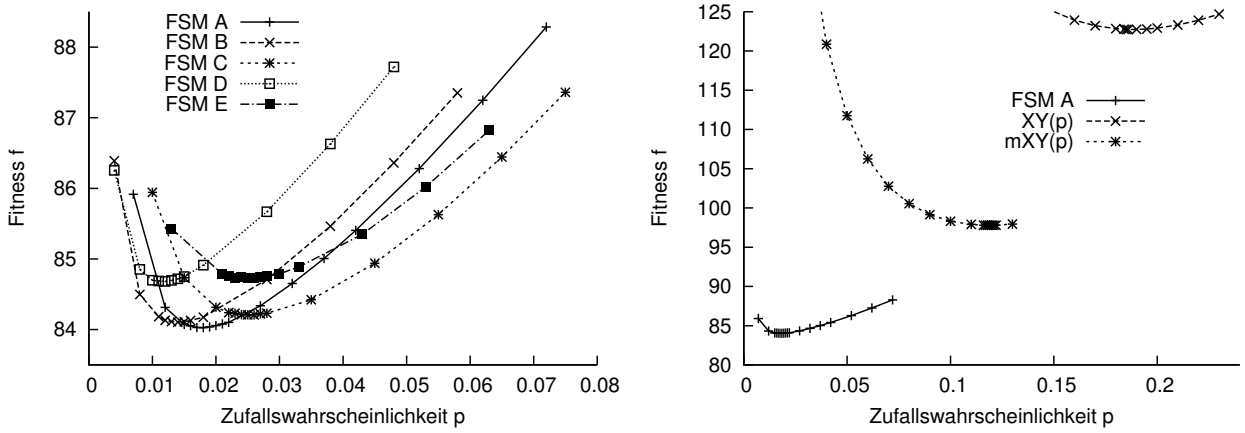

Abb. 6.19.: Fitnesswerte für $k = 256$ Agenten in Abhängigkeit der Zufallswahrscheinlichkeit. (a) die evolvierten FSMs A bis E, (b) die beste FSM A im Vergleich mit den XY-Routing-Agenten.

Durch die iterative Optimierung von p kann eine Kurve der Fitness in Abhängigkeit der Zufallswahrscheinlichkeit entwickelt werden. In Abb. 6.19(a) sind die Kurven der nach der Anpassung besten fünf FSMs (A bis E) für den Fall $k = 256$ dargestellt. Alle Kurven haben einen ähnlichen Verlauf, aber unterschiedliche Minima. Im Vergleich mit den Kurven, die für die XY-

Routing-Agenten entwickelt wurden, sieht man eine verbesserte Fitness und eine reduzierte Zufallswahrscheinlichkeit (Abb. 6.19(b)). Dies gilt für alle 20 getesteten FSMs.

Im letzten Abschnitt wurde für bestimmte Fälle und nicht randomisierte Agenten festgestellt, dass ein optimales Verhältnis von leeren Zellen zur Anzahl der Agenten zwischen 1 und 2 liegt. Weitere Simulationen mit der besten randomisierten FSM und je 1000 Welten der Größe $(N+z)\times(N+z)$ ($z \in \{1,2,\ldots,14\}$) mit 256 Agenten wurden durchgeführt. Die Start- und Zielpositionen wurden zufällig gewählt. Für $z = 10$ ergab sich der beste durchschnittliche Fitnesswert ($f = 69,3$). Das Verhältnis der leeren Zellen zu Agenten ist in diesem Fall 1,64. Das passt zu den Ergebnissen aus Abschn. 6.3.1.

Ergebnisse. Randomisierte FSMs lassen sich mit GA gut evolvieren. Allerdings kann durch nachträgliche Anpassung der Zufallswahrscheinlichkeit die Fitness noch gesteigert werden. Alternativ könnte der GA so modifiziert werden, dass die Fitnessbewertung zuverlässiger wird, indem schon während der Evolvierung mehrere Simulationen pro Lösungskandidat gemacht werden. Allerdings würde dann der GA erheblich langsamer. Es besteht ein Zusammenhang zwischen der Fitness und der Häufigkeit der zufälligen Entscheidungen der Agenten. Bessere evolvierte Agenten benötigen tendenziell weniger zufällige Entscheidungen.

6.4 Weitere Experimente und Ergebnisse

In diesem Abschnitt werden weitere Ergebnisse von Untersuchungen, die im Rahmen der Erstellung dieser Arbeit stattgefunden haben, präsentiert. Auf die detaillierte Beschreibung der Untersuchungen wird verzichtet.

6.4.1 Zyklischer Wrap-Around und Welten mit Rand im ATAC

In den Veröffentlichungen [EH08b] und [HE08] wird je ein Experiment beschrieben, in welchem Agenten mittels eines Inselmodell-GA evolviert werden, um den ATAC in Welten mit Wrap-Around und in Welten mit Rand zu lösen. Außerdem wurden die Lösungen der FSM-kontrollierten Agenten mit *Random Walkern* verglichen. Die Random Walker sind Agenten, die nicht durch eine FSM gesteuert sind, sondern in jeder Generation zufällig eine Entscheidung treffen.

In [HE08] wurde die Grundvariante des ATAC gewählt. In [EH08b] wurden andere Kommunikationssituationen verwendet (Abb. 6.20). Statt eine Mediatorzelle zu fordern, muss sich hier ein Agent in der Frontzelle eines anderen Agenten befinden, damit letzterer die Information des ersteren aufnehmen kann. Es kann also anders als bei der Grundvariante hier dazu kommen, dass die Informationen nur in einer Richtung ausgetauscht werden.

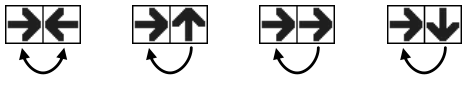

Abb. 6.20.: Alternative Kommunikationssituationen im ATAC. Die Pfeile geben den Informationsfluss an.

In beiden Experimenten haben die FSM-kontrollierten Agenten den ATAC wesentlich schneller gelöst als die Random Walker. Bemerkenswert ist die Tatsache, dass wie auch in Abschn. 6.2.1

festgestellt wurde, die Agenten in Welten mit Rand besser funktionieren als in Welten mit Wrap-Around, aber die Random Walker gerade umgekehrt auf Welten mit Wrap-Around schneller sind. Die Schlussfolgerung ist, dass der Rand für die sich zufällig bewegenden Agenten ein Hindernis darstellt, das mögliche Kommunikationssituationen verhindert und im Fall der FSM-kontrollierten Agenten eine hilfreiche Orientierung bietet.

Die Fitness während der Evolvierung wurde durch Simulation auf 10 Welten der Größe 33×33 (5 mit Rand, 5 mit Wrap-Around) bestimmt. Die so gefundenen besten 100 FSMs wurden dann auf einem größeren Set von Welten mit unterschiedlichen Größen (je 20 von 1089×1, 363×3, 121×9, 99×11) simuliert, um deren Robustheit gegenüber Veränderungen der Weltgröße festzustellen. Von den 100 FSMs waren alle in mindestens 37 der 80 neuen Welten erfolgreich. Nur eine einzige konnte alle 80 lösen. Am schwierigsten zu lösen waren die Welten der Größe 1089×1.

6.4.2 Gerichtete und ungerichtete Agenten im ARP

In diesem Abschnitt wird ein Experiment vorgestellt, bei dem es darum geht, die Effektivität von gerichteten und ungerichteten Agenten im ARP in verschieden großen Welten zu vergleichen. Diese Untersuchung wurde in [EH09b] beschrieben und ausgewertet. In dem Experiment wurden zwei Varianten des ARP untersucht und verglichen. Die eine (Fall A) ist die Grundvariante, wie in Abschn. 4.3 beschrieben. Die andere Variante (Fall B) hat ungerichtete Agenten, d. h. es fällt das Richtungsattribut weg und demzufolge auch die Aktionen, die die Richtung des Agenten modifizieren. Stattdessen kann der Agent aus 24 Entscheidungsmöglichkeiten auswählen ($e \in \{e_0, \ldots, e_{23}\}$). Jede der Entscheidungen entspricht einer der 24 möglichen Permutationsreihenfolgen der vier Himmelsrichtungen. Diese wird vom Aktuator durch ein Aktionsmapping in eine der fünf Aktionen $a \in \{N, E, S, W, C\}$ umgewandelt, mit der Bedeutung, sich auf die Nachbarzelle im Norden (N), im Osten (E), im Süden (S) oder im Westen (W) zu bewegen oder auf der momentanen Position zu verweilen (C). Der Aktuator bildet die Entscheidungen so ab, dass diejenige Bewegung mit der höchsten Priorität, die keinen Konflikt auslöst, ausgewählt wird. Das bedeutet auch, dass bei den ungerichteten Agenten die Prioritätenliste in der Zelle und die Möglichkeit eines Swap wegfällt. Eine weitere Änderung ergibt sich für die Inputreduktionstabelle (Tab. 6.18), da die eigene Richtung dort ja nicht mehr einfließen kann. Stattdessen wird das Zellfeld in neun Zonen eingeteilt, die absolut den Himmelsrichtungen zugeordnet werden können (Abb. 6.21).

Tab. 6.18.: Inputreduktionstabelle für das ARP mit ungerichteten Agenten.

Δx	<0	<0	<0	$=0$	$=0$	$=0$	>0	>0	>0
Δy	<0	$=0$	>0	<0	$=0$	>0	<0	$=0$	>0
x_r	0	1	2	3	4	5	6	7	8

Zum evolvieren der FSMs wurden neun verschiedene Sets von initialen Konfigurationen verwendet. Alle bestehen aus initialen Konfigurationen mit 32×32 Zellen ohne Hindernisse und einem zusätzlichen Rand. Die Sets unterscheiden sich in der Anzahl der Agenten $k \in \{1, 2, 4, 8, 16, 32, 64, 128, 256\}$. Die initialen Konfigurationen wurden entworfen, indem die k Agenten zufällig platziert worden sind (im Fall A auch mit zufälliger Richtung) und deren

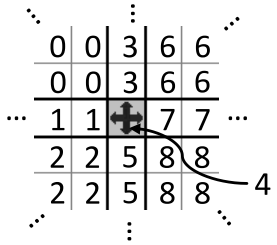

Abb. 6.21.: Zielsegmente im ARP für ungerichtete Agenten.

Zielposition ebenfalls zufällig gewählt wurde. Eine Einschränkung dabei war, dass keine zwei Zielpositionen identisch sind.

Jedes der neun Sets und jeder der zwei Fälle wurde separat mit einem Inselmodell-GA optimiert. Der Inselmodell-GA aus Abschn. 5.4.3 wurde hier verwendet, mit der Mutationstechnik $SG_{0,09}$, mit $I = 3$ Populationen mit je $|P| = 100$ Individuen und einer Elternquote von $q = 0,1$. Die Migrationswahrscheinlichkeit betrug $p = 0,02$. Der GA arbeitet in Fall A auf einem Suchraum von 24^{36} Kandidaten und in Fall B auf einem durch die hohe Anzahl der Outputs der FSM deutlich größeren Suchraum von 144^{54} Kandidaten. Die Fitnessfunktion ist eine Dominanzrelation, bei der folgende Größen in dieser Reihenfolge berücksichtigt werden:

1. Die Anzahl der Welten, in der die Agenten alle Ziele innerhalb von 10.000 Generationen erreichen, d. h. die Anzahl der erfolgreich gelösten Welten E.
2. Die Summe $\sum_{j=1}^{20} \bar{t}_j$ der nicht erreichten Zielpositionen nach 10.000 Generationen.
3. Die mittlere Dauer G wie in Abschn. 6.1.1, die nur für $E = 20$ definiert ist.

Für komplett erfolgreiche Lösungen (die in jeder Welt alle Zielpositionen erreichen) ist die Fitness äquivalent zur Anzahl der Generationen, die sie dafür benötigen. Zusätzlich zu den evolvierten Agenten wurden für beide Fälle auch MAS mit randomisierten Agenten simuliert, die nicht FSM-kontrolliert sind, sondern sich immer in Richtung der Zielposition bewegen bzw. drehen. Wenn es mehrere Möglichkeiten gibt, sich dem Ziel zu nähern, wird eine davon zufällig ausgewählt. In einer Welt ohne Konflikte bewegen sich diese Agenten immer optimal, d. h. auf kürzestem Weg.

Abb. 6.22 zeigt die Ergebnisse der besten gefundenen Lösungen und das theoretisch erreichbare Optimum, welches durch den Durchschnitt aller 20 maximalen Manhattan-Distanzen zwischen Start- und Zielposition der Agenten gegeben ist. Die randomisierten ungerichteten Agenten sind nur für Welten mit 8 oder weniger Agenten komplett erfolgreich, randomisierte gerichtete Agenten und FSM-kontrollierte ungerichtete Agenten für Welten mit 16 oder weniger Agenten. Nur die gerichteten FSM-kontrollierten Agenten können alle Welten lösen.

Bei den Agenten, die nicht erfolgreich waren, hat sich gezeigt, dass oft Deadlocks und Livelocks die Ursache waren. Bei den FSM-kontrollierten Agenten liegt es am Verhalten, das die FSM induziert. Bei den randomisierten Agenten liegt es daran, dass in einigen Fällen keine Auswahl aus mehreren möglichen kürzesten Wegen besteht und sich mehrere Agenten dann gegenseitig den Weg versperren.

Bei weiteren Tests zeigte sich außerdem, dass FSMs, die auf Welten mit vielen Agenten evolviert worden sind, ebenfalls auf Welten mit weniger Agenten erfolgreich sind, aber nicht so schnell wie die darauf evolvierten FSMs. Umgekehrt gilt der Zusammenhang nicht. FSMs, die für Welten mit wenigen Agenten optimiert worden sind, sind auf Welten mit vielen Agenten

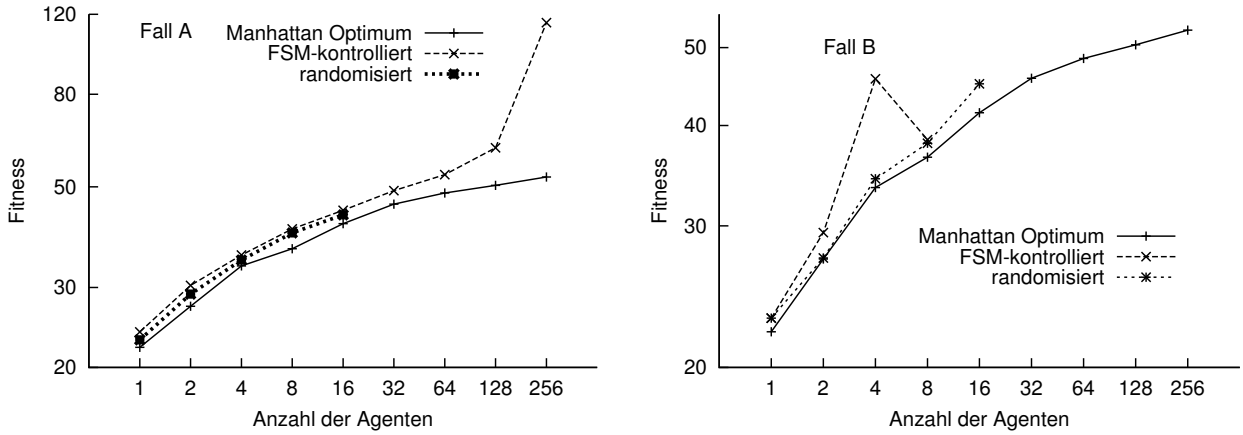

Abb. 6.22.: Fitness der gerichteten und ungerichteten Agenten nach Anzahl der Agenten im ARP.

nicht erfolgreich. Sie sind offenbar nicht dafür geeignet mit Situationen, in denen sich Agenten auf engem Raum stauen, umzugehen. In diesem Zusammenhang wurden einige Simulationen auf per Hand erstellten Welten durchgeführt, um die Strategie der Agenten zu analysieren. Abb. 6.23 zeigt zwei Beispielsequenzen von evolvierten FSM-kontrollierten Agenten. Die schnellere Strategie ist es, die Staus zu vermeiden, indem alle Agenten auf der gleichen Seite in Richtung Ziel laufen, langsamer ist der direkte und kürzere Weg, weil sich in der Mitte des Zellfeldes viele Konflikte ergeben.

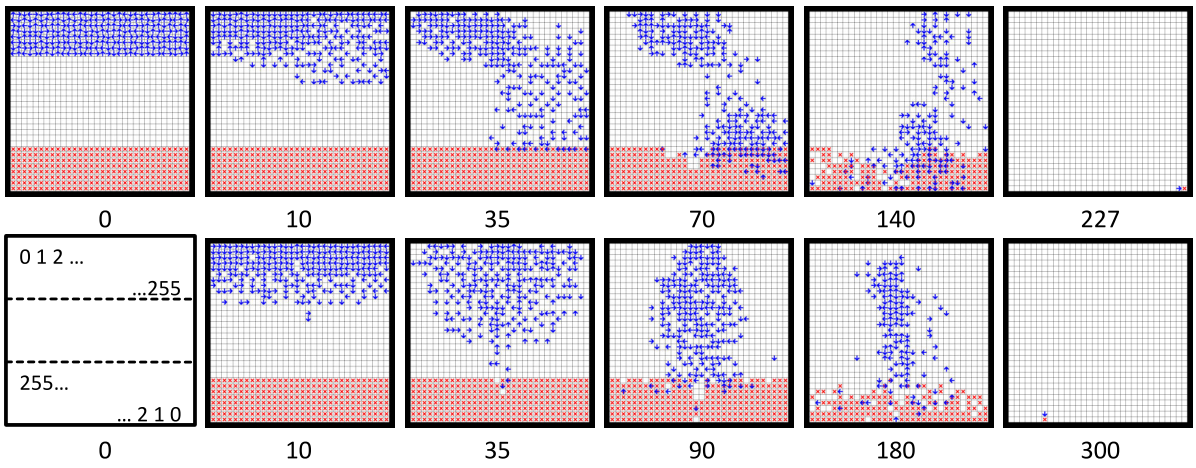

Abb. 6.23.: Beispielsequenz von FSM-kontrollierten Agenten im ARP. (a) Stau vermeidende Agenten, (b) den direkten kurzen Weg gehende Agenten. Die Zahlen links unten zeigen die initiale Verteilung der Positionen der Agenten und deren Ziele an.

6.4.3 Veränderung der Topologie für das ARP

In einem Experiment, das in [EHD10] beschrieben ist, wird das ARP auf einer Zellstruktur mit hexagonaler Topologie ausgeführt. Die Daten über die Qualität der Router (Agenten bzw. FSMs) können mit Daten auf einer orthogonalen Topologie verglichen werden. Die Anordnung und Verbindung der Zellen für beide Topologien ist in Abb. 6.24 für eine Weltgröße von 4×4 Zellen

dargestellt (auch bei der hexagonalen Topologie können zweidimensionale kartesische Koordinaten verwendet werden).

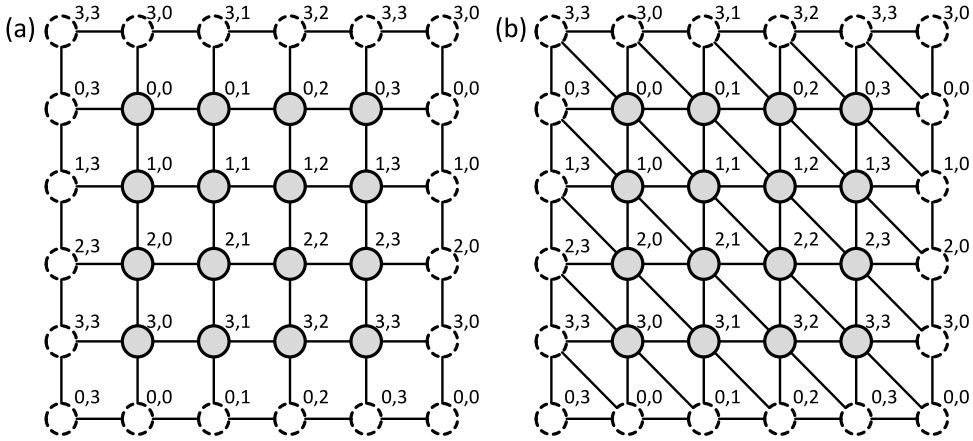

Abb. 6.24.: (a) Orthogonale und (b) hexagonale Verbindungsstruktur mit Wrap-Around.

Die Geschwindigkeit, mit der die Agenten ihr Ziel erreichen, hängt wesentlich davon ab, welchen Durchmesser und welche mittlere Distanz die Netzwerke (Zellfelder) haben. Der Durchmesser gibt die kürzeste Distanz zwischen den zwei am weitesten voneinander entfernten Knoten (Zellen) an, die mittlere Distanz ist der Durchschnitt der Distanzen aller Paare von Knoten. Sei eine zyklische Welt mit Wrap-Around der Größe $N \times N$ gegeben, wobei N zur Vereinfachung der folgenden Formeln eine Zweierpotenz sei, dann ist in orthogonalen Topologien der Durchmesser $D_o = N$ und die mittlere Distanz $\delta_o = N/2$. Die Formeln für die hexagonale Topologie sind etwas komplexer und ergeben approximiert: $D_h \approx 2N/3$ bzw. $\delta_h \approx (7N^2 - 3)/(18N)$. Das Verhältnis der Durchmesser ist $D_o/D_h \approx 1,5$ und das der mittleren Distanzen ist $\delta_o/\delta_h \approx 1,29$.

Die Fitness der Agenten in erfolgreich gelösten Welten wird durch den langsamsten Agenten definiert. Dies ist bei einer hohen Dichte sehr wahrscheinlich einer, der den Durchmesser oder annähernd den Durchmesser bis zu seiner Zielposition zurücklegen muss. Bei einem Wechsel der Topologie von orthogonal auf hexagonal unter Beibehaltung der gleichen Anzahl an Zellen kann man also eine Verbesserung der Fitness um Faktor $1,5$ erwarten.

Eine Umstellung auf eine hexagonale Topologie bedeutet aber auch höhere Kosten für die Einrichtung eines solchen Routers. Zum einen sind 1,5mal so viele Verbindungen notwendig, zum Anderen müssen die Agenten angepasst werden. Aus neun möglichen Zielzonen werden zwölf. Aus vier Richtungen werden sechs, dementsprechend mehr Drehaktionen etc. Der Aufwand für diese Erweiterungen wird ebenfalls mit etwa Faktor 1,5 überschlagen.

In der Untersuchung stellte sich heraus, dass der Faktor, um den die Fitness bei gleichbleibender Komplexität der FSM verbessert werden konnte, in Abhängigkeit von der Größe der Welt, der Dichte der Agenten darin und der Verwendung von randomisierten FSMs zwischen 1,18 und 1,91 betrug. Insbesondere für große Welten mit einer hohen Dichte von Agenten, war dieser Faktor hoch.

6.4.4 Das Particle Alignment Problem

Ein weiteres MAS, das im CA-Modell aus dieser Arbeit definiert wurde, ist das *Particle Alignment Problem* (PAP). Dieses stellt eine Erweiterung des in eindimensionalen CA mit zwei Zuständen

nicht lösbaren Majority-Problems [LB95] dar. Die Agenten können dabei die Zellen in einer von vier zur Verfügung stehenden Farben färben. Das globale Problem ist, erstens alle Zellen mit der gleichen Farbe zu färben und zweitens dafür die Farbe zu wählen, in der in der initialen Konfiguration die Mehrheit der Zellen gefärbt ist. Der Name Particle Alignment Problem rührt daher, dass jede Farbe auch als Richtung (z. B. eines Partikels, das sich auf der Zelle befindet) uminterpretiert werden kann. Dann wäre die Aufgabe der Agenten nicht eine Gleichfärbung, sondern eine Gleichausrichtung aller Zellen bzw. Partikel.

Das Problem wurde in [EH09a] beschrieben und untersucht. Dabei stellte sich heraus, dass die Einfärbung des gesamten Zellfeldes in irgendeiner Farbe für die Agenten zu bewältigen war, nicht aber die Einfärbung in der richtigen Farbe. Darüber hinaus wurde teilweise ein oszillierendes Verhalten des Systems beobachtet, d. h. die Farbe oder Ausrichtung des gesamten Feldes wurde zyklisch gewechselt (Abb. 6.25). Das PAP wird in dieser Arbeit nicht weiter untersucht, weil die Ergebnisse suggerieren, dass das Problem in dieser Variante nicht lösbar ist.

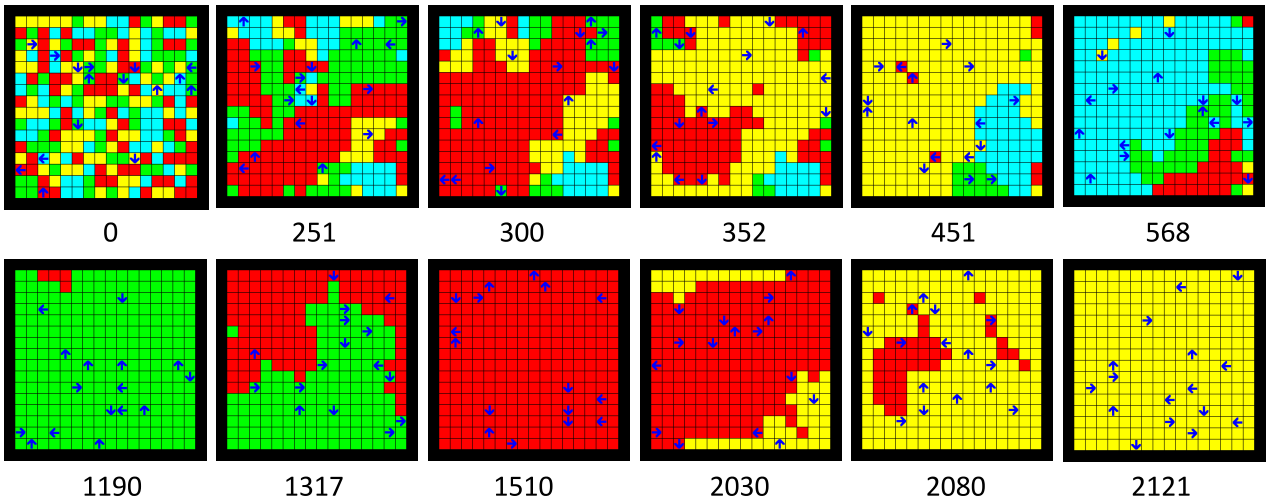

Abb. 6.25.: Beispielsequenz eines oszillierenden Verhaltens im PAP.

6.5 Kapitelzusammenfassung

Die Optimierungstechniken, die die Struktur der Kontrolleinheit der Agenten anpassen und Anpassungen der Agentenwelt wurden zusammen mit Genetischen Algorithmen für die Beispielsysteme CEP, ATAC und ARP eingesetzt und die Ergebnisse analysiert.

Im CEP konnte die Effizienz der Agenten durch Variation der Anzahl der Agenten gesteigert werden. Die Verwendung von heterogenen Agententypen in einer Welt brachte nur bei geringer Anzahl von Agenten einen Vorteil. Time-Shuffling war sehr effektiv, aber nur für TS-Perioden in einem bestimmten Intervall.

Die Verwendung eines Randes lieferte für die Agenten im ATAC eine Orientierungshilfe. MAS mit Time-Shuffling werden am besten gemeinsam optimiert. Die Erweiterung der Agentenfähigkeiten ist zum Teil effektiv. Der GA ist dabei bis zur getesteten Komplexität zuverlässig. Aber nicht alle Fähigkeiten bringen einen Vorteil (z. B. das Acknowledge-Signal beim ATAC).

Durch Randomisierung der Entscheidung der Agenten konnten Deadlock-Situationen im ARP aufgelöst werden. Der Durchsatz von Nachrichten kann durch die Anpassung der Größe der Welt bzw. dem Verhältnis der Agentenanzahl und der freien Zellen gesteigert werden.

Der Genetische Algorithmus konnte mit unterschiedlichen Techniken und für verschiedene Suchräume unterschiedlicher Größe erfolgreich eingesetzt werden. Auch der inkrementelle GA lieferte akzeptable Lösungen.

7 Zusammenfassung und Ausblick

Zusammenfassung. Die Ziele dieser Arbeit waren:

- Der Entwurf eines Modells zur Beschreibung von Multiagentensystemen im Zellularen Automaten, welches die Einbindung von beliebigen Strukturen zur Programmierung des Agentenverhaltens erlaubt.
- Die Entwicklung/Wahl eines geeigneten heuristischen Verfahrens (in Kombination mit einer geeigneten strukturellen Beschreibung für das Agentenverhalten), um das Agentenverhalten zu optimieren, sowie die Entwicklung von Techniken zur (problemspezifischen) Anpassung der Struktur des Agentenverhaltens.

Unter Berücksichtigung diverser Vorarbeiten wurde ein allgemeines Modell für Multiagentensysteme in Zellularen Automaten entworfen. Dieses Modell erlaubt eine beliebige Implementierung einer Kontrollfunktion für das Agentenverhalten. Für die Konfliktauflösung zwischen sich bewegenden Agenten und die Ermöglichung von Modifikationen auf Nachbarzellen werden erweiterte Nachbarschaften und konsistente Regeln verwendet, die einen Schreibzugriff emulieren. Durch diese Emulation können auch interzellulare Bewegungen von Agenten, oder Objekten allgmein, realisiert werden. Das erste Ziel wurde damit erreicht.

Das Modell wurde verwendet, um drei Beispielsysteme (Creatures' Exploration Problem, All-to-All Communication Task und Agent Routing Problem) und einige Varianten davon zu implementieren. Dabei wurden die Kontrollfunktionen der Agenten mit endlichen Zustandsautomaten implementiert. Um die Komplexität der Struktur der Zellen, also deren Bedarf an Hardware-Ressourcen, gering zu halten, wurde das allgemeine Modell für jedes der Beispielsysteme manuell optimiert, indem redundante oder ungenutzte Ressourcen weggelassen oder zusammengefasst wurden. Die Beispiele sind eine Bestätigung für die Einsetzbarkeit und Flexibilität des Modells.

Die globalen Zielvorgaben für die Agenten können stark variieren und die Wahl einer Methode zur Verhaltensoptimierung einschränken. In dieser Arbeit sind Maßnahmen zur Optimierung durch Veränderung der Agentenwelt und Techniken zur Optimierung durch Anpassung der Struktur der Kontrolleinheit eines Agenten vorgestellt worden. Der Inhalt der Kontrollfunktion der Agenten wurde mit einem Genetischen Algorithmus optimiert.

Die Effektivität des Genetischen Algorithmus zeigt sich in verschiedenen Anwendungsbeispielen und Konfigurationen. Es lässt sich daraus schließen, dass der Genetische Algorithmus zuverlässig ist. Die Zuverlässigkeit wurde im Beispiel des Creatures' Exploration Problem auch gegenüber eines bekannten Optimums verifiziert, auch bei verschiedenen Variationen der Parameter. Die Größe des Suchraums war dabei begrenzt. Ob der Genetische Algorithmus auch bei deutlich komplexeren Kontrollfunktionen effektiv ist, wurde nicht getestet. Gegenüber der kompletten Suche des gesamten Suchraums ist der Genetische Algorithmus jedoch eine deutliche Verbesserung. Damit wurde ein geeignetes heuristisches Verfahren (bezogen auf endliche Zustandsautomaten) gefunden.

Zur problemspezifischen Anpassung der Agentenstruktur wurden mehrere Techniken entwickelt. *Randomisierte FSMs* können eingesetzt werden, wenn die Zielvorgabe der Agenten so

gestaltet ist, dass ohne Zufallskomponente Deadlocks oder Livelocks unvermeidlich oder zumindest wahrscheinlich sind. Die *Time-Shuffling-Technik* und die Verwendung von *heterogenen Agententypen* können für Problemstellungen eingesetzt werden, in denen jeweils für unterschiedliche Teilprobleme unterschiedliche Strategien (Verhaltensmuster) sinnvoll sind. Hier ist allerdings eine richtige Auswahl der einzelnen zu kombinierenden FSMs (und damit die durch die FSMs induzierten Strategien) notwendig. Die besten Ergebnisse konnten mit Kombinationen von FSMs erzielt werden, die für sich alleine die Probleme oder Teilprobleme nicht gut gelöst werden.

Komplexe Verhaltensmuster können schon mit relativ wenigen (zwischen fünf und acht) Zuständen der endlichen Automaten erzeugt werden. Diese können mit dem Genetischen Algorithmus gut evolviert werden. Der Genetische Algorithmus kann teilweise ein Verhalten evolvieren, das im Nachhinein Sinn ergibt, aber dessen Semantik im Vorhinein sehr schwierig von Hand zu entwickeln gewesen wäre (z. B. synchronisierendes Verhalten oder die Bildung von Straßen mit Flags). Die *Erweiterung der Agentenfähigkeiten* erhöht die Komplexität der Agenten und den Suchraum für den Genetischen Algorithmus exponentiell. Dennoch ist der Genetische Algorithmus auch hier im getesteten Rahmen zuverlässig und findet bessere Lösungen.

Einige der vorgeschlagenen Techniken waren auf den getesteten Agentenwelten effektiv. Diese Effektivität kann aber nicht immer verallgemeinert werden. Beim Time-Shuffling zum Beispiel hängt die Effektivität von mehreren Parametern, wie der Time-Shuffle-Periode und der Anzahl der Agenten in der Agentenwelt, ab, aber auch von der Optimierungsmethode und letztlich dem globalen Ziel (der Beispielanwendung) der Agenten.

Ausblick. Einige Fragen bezüglich der Optimierung des Agentenverhaltens sind offen geblieben. Es wurde im Wesentlichen nur der Genetische Algorithmus angewendet. Es wurde gezeigt, dass andere Verfahren, wie Genetische Programmierung, Q-Learning, Ant Colony Optimization und weitere, grundsätzlich anwendbar sind. Weitere Tests könnten nun klären, ob diese ebenfalls akzeptable oder sogar bessere Ergebnisse liefern.

Die Komplexität der Struktur der Kontrolleinheit der Agenten wurde in dieser Arbeit durch die Beschränkung der Inputs, Outputs und Kontrollzustände relativ klein gehalten. Eine Frage, die in zukünftigen Arbeiten beantwortet werden könnte, ist, ob die Optimierungstechniken und heuristischen Verfahren auch bei deutlich komplexeren Agenten akzeptable Ergebnisse liefern. Eine Erhöhung der Komplexität kann auch dadurch hergestellt werden, dass der Bereich der Kontrolleinheit, der heuristisch optimiert wird, vergrößert wird. So kann z. B. eine Inputreduktion in das Genom integriert werden.

Für das Modell für Multiagentensysteme in Zellularen Automaten sind ebenfalls Erweiterungen denkbar. Teilweise bereits eingesetzt wird z. B. das Modell des Globalen Zellularen Automaten, das dasjenige des Zellularen Automaten um dynamische und globale Nachbarschaften erweitert. Die Anwendbarkeit und Effektivität der Optimierungstechniken und heuristischen Verfahren könnten auch im Globalen Zellularen Automaten untersucht werden.

A Quellcodes

List. A.1 ist die Verilog-HDL-Beschreibung einer Zelle im Creatures' Exploration Problem (Abschn. 4.1). Mit einem geeigneten Programm kann dieser Code für bestimmte Bausteine (Field Programmable Gate Arrays) synthetisiert werden. Die so entstehende Hardware kann von der in Abb. 4.2 dargestellten abweichen.

Das Listing enthält zwei Module: *FSMtable* und *CEPcell*. Letzteres verwendet das andere Modul in einer Instanz und stellt die gesamte Zellstruktur dar. Es bekommt die Zellzustände der Nachbarn und ein Taktsignal als Inputs. Mit mehreren *case*- und *assign*-Anweisungen wird die kombinatorische Logik realisiert. Die Zuweisungen der neuen Zustände ist an den Takt gebunden. Für die Zuweisungen und Vergleiche der Zellypen, Richtungen und Entscheidungen werden Parameter verwendet. Weder die Zustände der Zelle noch die Tabelle der FSM sind im Programmcode initialisiert.

Eine Synthese mit Altera Quartus II 8.1 ergibt die Nutzung von 27 Logikelementen (Adaptive Lookup Tables). Für ein ganzes Zellulares Feld muss diese Anzahl vervielfacht werden und zusätzlich der Aufwand für die Verdrahtung der Nachbarzellen hinzugerechnet werden.

List. A.1: Beschreibung der Hardware für eine Zelle im CEP in Verilog

```
1  module CEPcell (t_n, t_s, t_e, t_w, t_nn, t_ne, t_nw, t_se, t_sw, t_ss, t_ee,
       t_ww, d_n, d_s, d_e, d_w, d_nn, d_ne, d_nw, d_se, d_sw, d_ss, d_ee, d_ww
      , s_n, s_s, s_e, s_w, clk);
2
3  parameter N=2'd0, E=2'd1, S=2'd2, W=2'd3;
4  parameter Leer=2'd0, Creature=2'd1, Hindernis=2'd2;
5  parameter L=1'b0, R=1'b1;
6
7  input clk;
8  input[1:0] t_n, t_s, t_e, t_w, t_nn, t_ne, t_nw, t_se, t_sw, t_ss, t_ee, t_ww
       , d_n, d_s, d_e, d_w, d_nn, d_ne, d_nw, d_se, d_sw, d_ss, d_ee, d_ww;
9  input[2:0] s_n, s_s, s_e, s_w;
10 reg[2:0] s;
11 reg [1:0] t_f, d_1, d_2, d_3, d_4, t_1, t_2, t_3, t_4, d_new, t, d;
12 wire [1:0] t_new, h_mL, d_MUX;
13 wire [2:0] s_new, s_MUX;
14 wire h_S_out, h_E_out, h_N_out, h_W_out, h_1_out, h_2_out, h_3_out, h_4_out,
       m, m_L, e;
15
16 FSMtable fsm(.s_in(s_MUX), .m(m), .s_out(s_new), .e(e));
17
18 always@*
19 begin
20 case (d)
21   N: begin t_f = t_n; t_1 = t_nn; d_1 = d_nn; t_2 = t_ne; d_2 = d_ne; t_3 =
         t_nw; d_3 = d_nw; t_4 = Leer; d_4 = N; end
22   E: begin t_f = t_e; t_1 = t_ne; d_1 = d_ne; t_2 = t_ee; d_2 = d_ee; t_3 =
         Leer; d_3 = E; t_4 = t_se; d_4 = d_se; end
23   S: begin t_f = t_s; t_1 = Leer; d_1 = S; t_2 = t_se; d_2 = d_se; t_3 = t_sw
         ; d_3 = d_sw; t_4 = t_ss; d_4 = d_ss; end
```

```
24   W: begin t_f = t_w; t_1 = t_nw; d_1 = d_nw; t_2 = Leer; d_2 = W; t_3 = t_ww
         ; d_3 = d_ww; t_4 = t_sw; d_4 = d_sw; end
25 endcase
26 case (e)
27   L: begin d_new = (d_MUX + 3); end
28   R: begin d_new = (d_MUX + 1); end
29 endcase
30 end
31
32 assign h_W_out = (W==d_e) & (t_e==Creature);
33 assign h_N_out = (N==d_s) & (t_s==Creature);
34 assign h_E_out = (E==d_w) & (t_w==Creature);
35 assign h_S_out = (S==d_n) & (t_n==Creature);
36 assign h_1_out = (S==d_1) & (t_1==Creature);
37 assign h_2_out = (W==d_2) & (t_2==Creature);
38 assign h_3_out = (E==d_3) & (t_3==Creature);
39 assign h_4_out = (N==d_4) & (t_4==Creature);
40 assign h_mC = ~(h_S_out|h_E_out|h_N_out|h_W_out);
41 assign h_mL = h_S_out+h_E_out+h_N_out+h_W_out;
42 assign m_L = h_mL==1'b1 & t==Leer;
43 assign m_C = h_mC & t_f==Leer;
44 assign m = (t==Creature)? m_C : m_L;
45 assign t_new = (t==Hindernis)? Hindernis : ((t==Creature)? (m? Leer :
       Creature) : (m? Creature : Leer) );
46 assign d_MUX = (h_W_out & m_L)? d_w : ((h_N_out & m_L)? d_n : ((h_E_out & m_L)
       ? d_e : ((h_S_out & m_L)? d_s : (d))));
47 assign s_MUX = (h_W_out & m_L)? s_w : ((h_N_out & m_L)? s_n : ((h_E_out & m_L)
       ? s_e : ((h_S_out & m_L)? s_s : (s))));
48
49 always @(posedge clk)
50 begin t ⇐ t_new; d ⇐ d_new; s ⇐ s_new; end
51 endmodule
52
53
54 module FSMtable(s_in,m,s_out,e);
55 input[2:0] s_in;
56 input m;
57 output[2:0] s_out;
58 output e;
59 reg[2:0] tabelle_s [15:0];
60 reg tabelle_e [15:0];
61 assign s_out = tabelle_s[{s_in, m}];
62 assign e = tabelle_e[{s_in, m}];
63 endmodule
```

B Initiale Konfigurationen

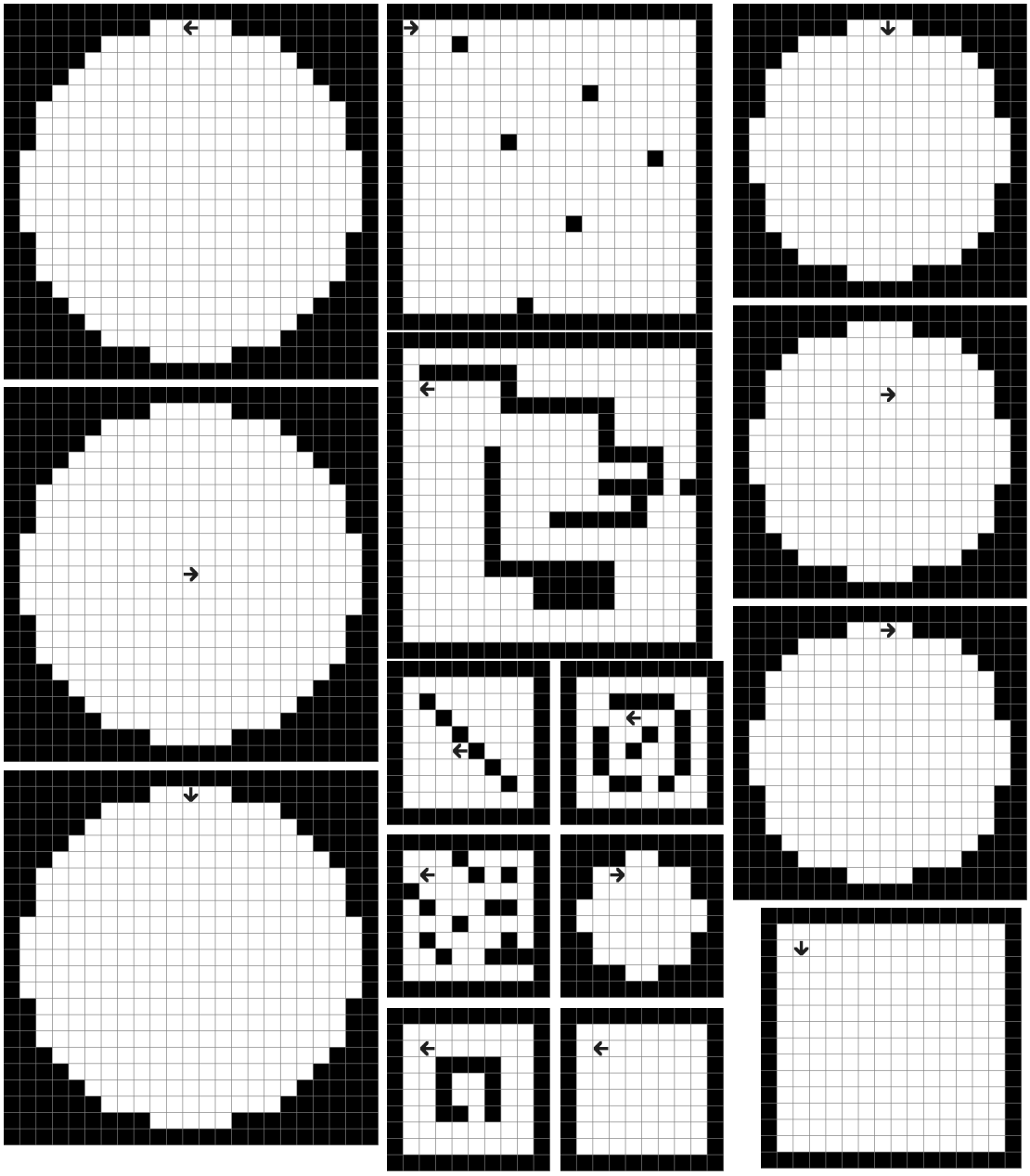

Abb. B.1.: 26 initiale Konfigurationen des CEP mit je einem Agenten - Teil 1 (wie in [Hal08]).

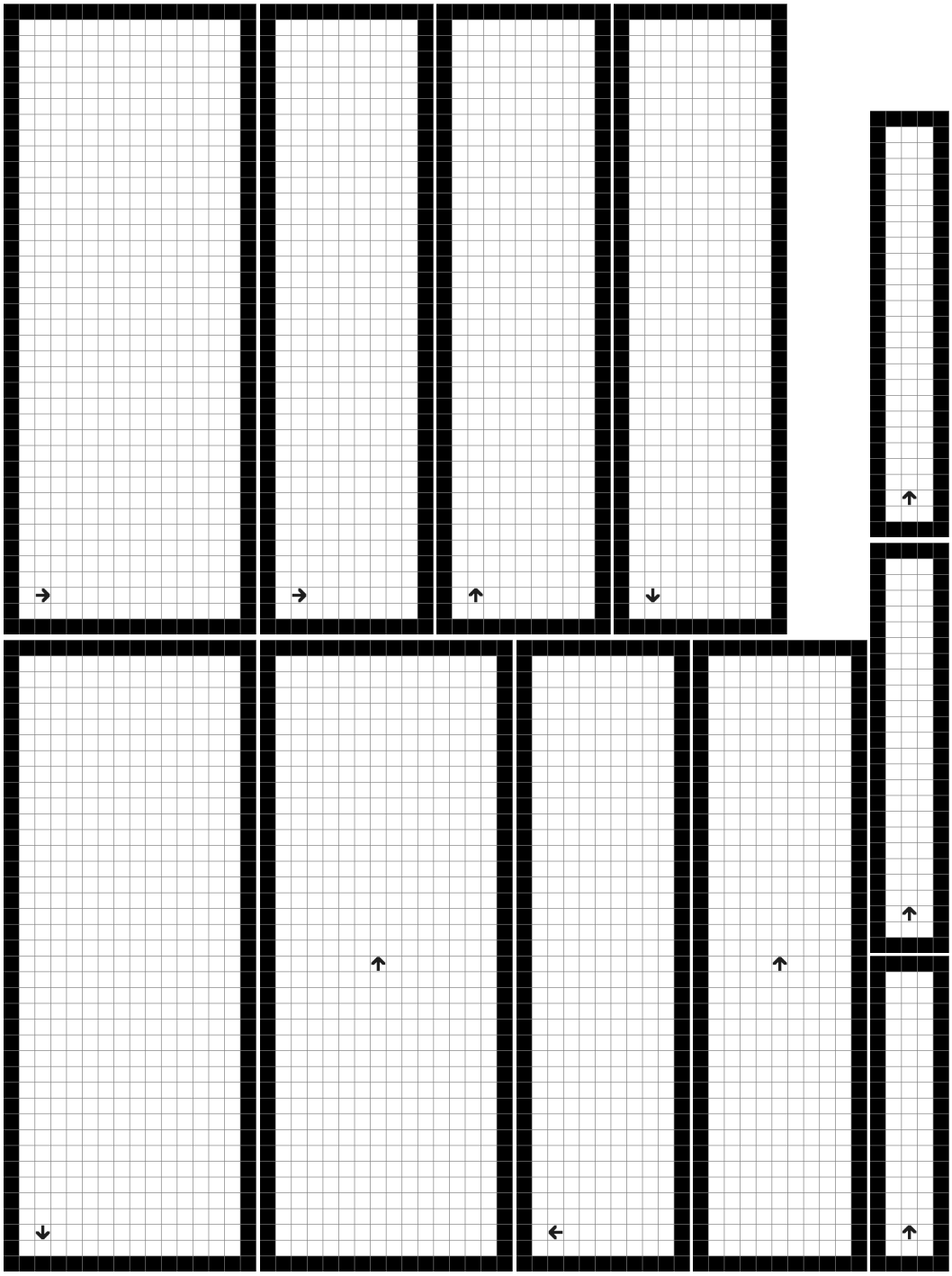

Abb. B.2.: 26 initiale Konfigurationen des CEP mit je einem Agenten - Teil 2 (wie in [Hal08]).

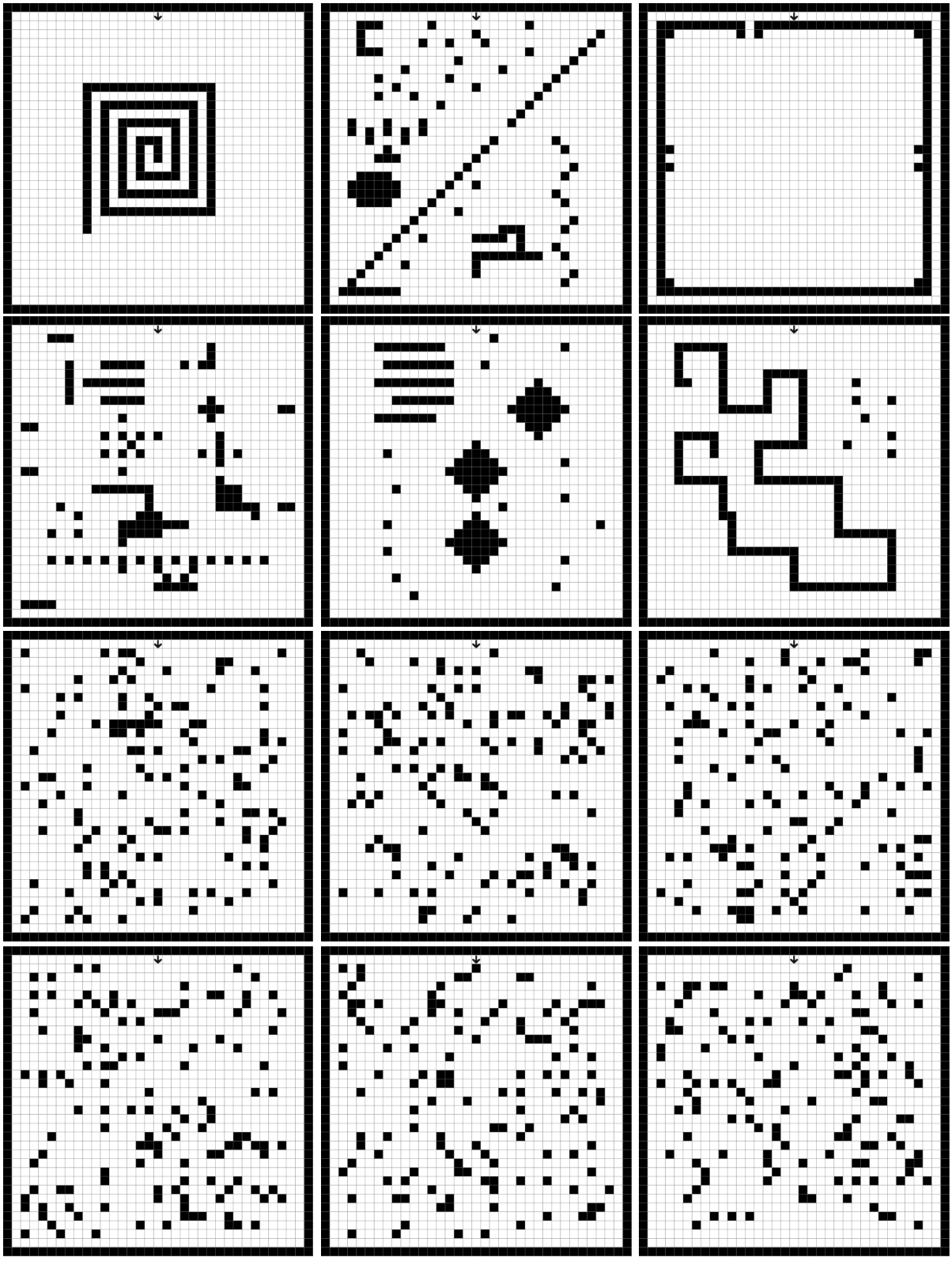

Abb. B.3.: 12 initiale Konfigurationen mit 33×33 Zellen (Rand unberücksichtigt), 960 freien Zellen sowie 129 Hindernissen und einem Agenten für die Experimente in Abschn. 6.1.

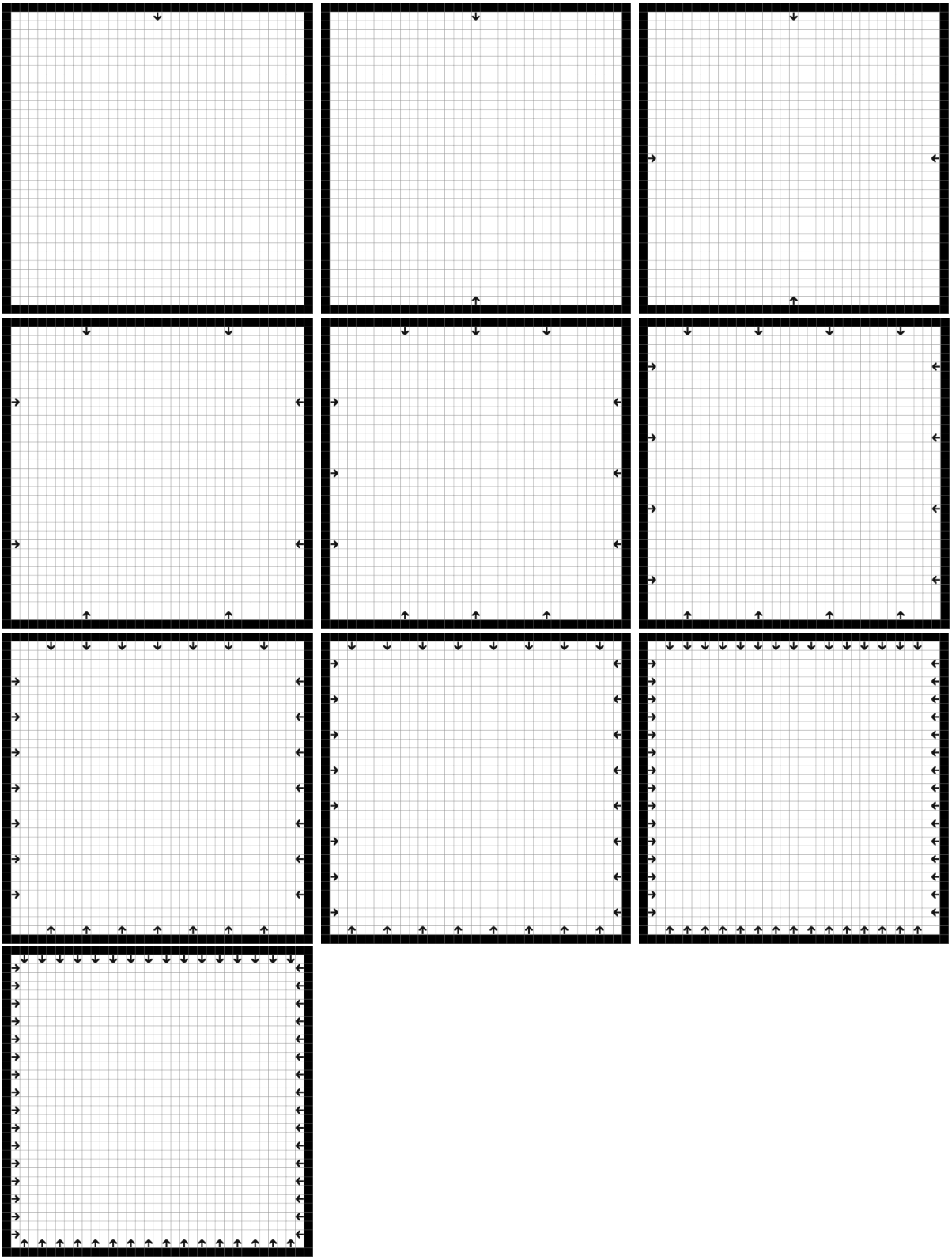

Abb. B.4.: Verteilung der 1, 2, 4, 8, 12, 16, 28, 32, 60 bis 64 Agenten auf den initialen Konfigurationen mit 33×33 Zellen (Rand unberücksichtigt) für die Experimente in Abschn. 6.1 (vgl. [Hal08]). Die Hindernisse sind nicht dargestellt.

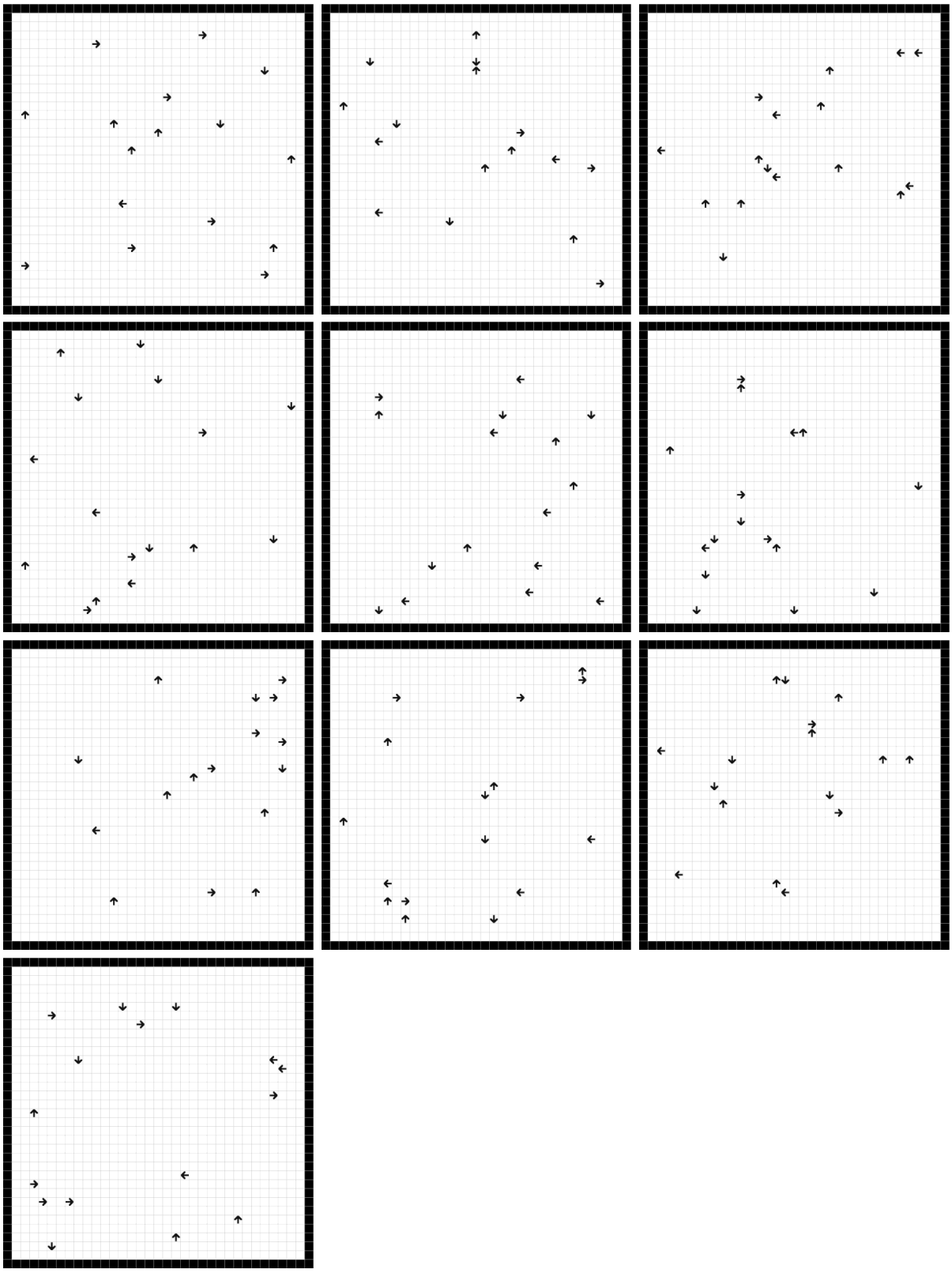

Abb. B.5.: 10 initiale Konfigurationen mit Rand und 33×33 Zellen (Rand unberücksichtigt) und 16 Agenten für die Experimente in Abschn. 6.2.1.

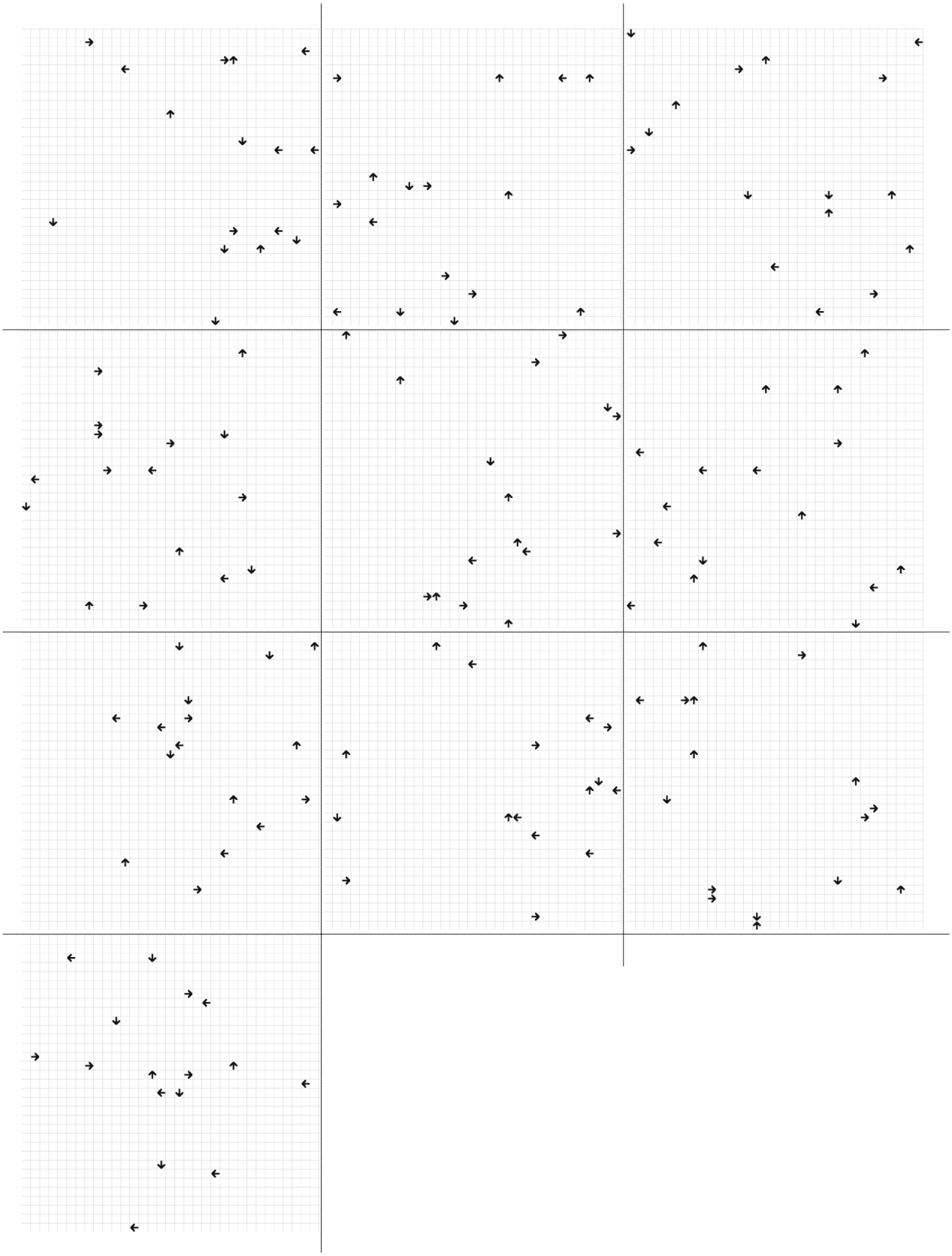

Abb. B.6.: 10 initiale Konfigurationen mit 33×33 Zellen ohne Rand und 16 Agenten für die Experimente in Abschn. 6.2.1.

C Listen interessanter evolvierter FSMs

In diesem Kapitel werden interessante evolvierte Automaten aufgelistet, die aus den Untersuchungen aus Kap. 5 und Kap. 6 hervorgegangen sind und mit denen sich die dort präsentierten Fitnessbewertungen nachvollziehen lassen.

Tab. C.1 enthält die 21 FSMs mit fünf Zuständen aus der Untersuchung aus Abschn. 5.4, die einen Fitnesswert von $f > 6000$ aufweisen.

Tab. C.1.: Die 21 erfolgreichsten FSMs mit fünf Zuständen aus der Untersuchung aus Abschn. 5.4, die einen Fitnesswert von $f > 6000$ aufweisen.

Genom	f
1/L 3/R 0/R 0/L 2/R - 2/L 1/R 0/R 4/R 3/L	6140
1/L 3/L 3/R 4/R 2/L - 2/R 3/L 0/L 1/R 4/L	6100
1/L 3/R 3/R 0/L 2/R - 2/L 1/R 0/R 4/R 3/L	6084
1/L 3/L 4/R 2/R 0/L - 2/L 2/R 0/R 4/L 3/R	6084
1/L 3/R 1/L 0/L 2/R - 2/L 1/R 0/R 4/R 3/L	6073
1/R 3/L 1/R 2/R 0/L - 2/L 1/L 0/R 4/L 3/R	6072
0/R 2/R 3/L 1/R 0/L - 1/L 0/R 2/L 4/L 3/R	6072
1/L 2/R 3/L 1/R 0/L - 1/L 0/R 2/L 4/L 3/R	6072
1/L 2/R 4/R 0/L 1/L - 2/L 3/L 0/R 1/R 4/L	6072
1/L 3/R 2/L 0/L 2/R - 2/L 1/R 0/R 4/R 3/L	6062
1/L 1/R 3/R 4/R 2/L - 2/R 3/L 0/L 1/R 4/L	6061
1/R 3/L 0/L 0/R 2/L - 2/R 1/L 0/L 4/L 3/R	6059
1/R 2/L 3/R 1/L 0/R - 1/R 0/L 2/R 4/R 3/L	6044
1/L 2/R 4/R 0/L 3/R - 2/L 3/L 0/R 1/R 0/L	6041
1/R 3/R 3/L 4/L 2/R - 2/L 3/R 0/R 1/L 4/R	6035
1/L 2/L 3/R 4/R 2/L - 2/R 3/L 0/L 1/R 4/L	6023
1/L 3/R 4/R 2/L 3/R - 2/R 4/L 0/L 3/L 1/R	6021
1/L 3/L 4/R 0/R 0/R - 2/R 4/L 0/L 3/R 1/R	6012
1/L 3/L 4/R 0/R 1/R - 2/R 4/L 0/L 3/R 1/R	6012
1/L 3/L 4/R 0/R 3/L - 2/R 4/L 0/L 3/R 1/R	6012
1/L 3/L 4/R 0/R 4/L - 2/R 4/L 0/L 3/R 1/R	6012

Tab. C.2 enthält die elf FSMs mit fünf Zuständen aus der Untersuchung aus Abschn. 5.4.3, die alle fünf Welten lösen können.

Die Tabellen C.3 bis C.8 zeigen die jeweils besten evolvierten Systeme und deren Fitness von jedem der 30 Durchläufe des GA für jeweils einen der evolvierten Typen Z, XY_T, UV_T-u, UV_T-m, U_TV_T-u oder U_TV_T-m aus Abschn. 6.2.1 an. Aus Platzgründen sind hier im Genom die Schrägstriche weggelassen. Aus den hier angegebenen Fitnesswerten ergibt sich die Verteilung, die in Abb. 6.4 dargestellt ist.

Tab. C.9 beinhaltet die für jede Entscheidungsmenge aus der Untersuchung in Abschn. 6.2.2 beste FSM, die in irgendeinem der 30 Durchläufe des GA gefunden wurde. Die Reihenfolge der aus sechs Zustandsübergangspaaren bestehenden Teile des Genoms (mit Bindestrich oder Zeilenumbruch getrennt) ergibt sich aus der Reihenfolge der Inputkombinationen, bestehend

Tab. C.2.: Die elf FSMs mit fünf Zuständen, die alle fünf Welten aus der Untersuchung aus Abschn. 5.4.3 lösen können.

Genom	f
1/L 3/L 2/R 0/L 4/R - 2/L 0/R 4/L 3/R 1/R	3284
1/R 3/R 2/L 0/R 4/L - 2/R 0/L 4/R 3/L 1/L	3566
0/L 1/L 3/R 4/R 2/R - 1/R 2/L 4/L 3/L 0/R	3857
0/R 1/R 3/L 4/L 2/L - 1/L 2/R 4/R 3/R 0/L	4352
1/L 2/L 0/L 3/R 4/R - 2/R 1/R 3/L 4/L 0/R	4920
1/R 2/R 0/R 3/L 4/L - 2/L 1/L 3/R 4/R 0/L	5154
0/R 2/L 1/L 3/L 4/R - 1/R 3/R 2/R 4/L 0/L	5462
1/L 0/L 2/R 3/R 4/L - 2/L 1/R 3/L 4/R 0/R	5891
1/L 0/L 3/R 2/R 4/L - 2/L 1/R 3/L 4/R 0/R	5974
0/R 2/L 3/L 1/L 4/R - 1/R 0/L 2/R 4/L 3/R	6270
0/L 2/R 3/R 1/R 4/L - 1/L 0/R 2/L 4/R 3/L	6445

aus dem Wert des Flags F und dem Konfliktanzeiger m. Zuerst sind die Übergänge für $(F = 0, m = 0)$, dann $(F = 0, m = 1)$, dann $(F = 1, m = 0)$ und schließlich $(F = 1, m = 1)$ angegeben. In den Fällen U und V fallen die letzten beiden Teile des Genoms weg, da kein Flag gesetzt werden kann.

Tab. C.3.: Die jeweils beste evolvierte FSM vom Typ Z aus jedem der 30 Durchläufe des GA für das Experiment aus Abschn. 6.2.1.

Genom	f
3L4R3L4R0R2R-3R4L1L5L0R2R	605,6
4R4R2R5R3L1R-5L2L0R1R3R4L	643,4
5L1R1R3L2R5L-3R5R1L2L0L4R	608,6
1R4L2L4L2L4L-3L4R5L1L2R0R	654,4
5L1L2R4R2R5L-1R4L5R0L2L3R	608,6
2L1R1R1R5R5R-2R4L1L5L3R0R	638,1
2L1L5R2L3L4L-4R0L5L1R2R3L	643,4
5R4L2L3R2L5R-3L0R5L4R2R1L	608,6
3R5L5L2R2L3L-4L5R0L2R1R3L	629,0
4R0L1R0L1L2L-2L5L1R0R3R4L	630,8
4R2R0L1L2L0L-1L2R3L4L5R0R	630,8
2R4R1L1R2R0R-3R4R1R5L0L2L	650,8
3R5L2R3R4L4L-2L0R5R1L3L4R	608,6
1R2L0L5L2L1R-1L3R0L5L2R4R	605,6
2R1R2R5L3L5L-1L3R4L5R0R2L	608,6

Genom	f
1R1R3L3L2L5R-5L4L3R1L0R2R	608,6
1R1R2R4L5L5L-2L3L4R0R5R1L	608,6
2R4L2R1L4L5R-5L4R3L0R2L1R	608,6
2L0R2L0R3L5L-2R5R4R0L1L3L	651,6
5R3L2L2L4R5R-4L0R5L2R3R1L	608,6
1R2L2L2L5L5L-1L2R4R5R3L0L	638,1
5L4L1L3L5L2R-1R5R4L0L3R2L	643,4
5R2L2L3R1L5R-3L2R5L1R0R4L	608,6
4R1L0L1L4R0L-5L2R3L0R1R4L	647,3
1R1R2L3R2L4L-3L5L1L4R2R0R	608,6
5R1L3L3L1L1L-5L2R4L0L3R1R	638,1
2R4L2R3R4L1L-3L4R5L1R2L0R	608,6
2R3L5L3L5L5L-2L4L5R0L3R1R	638,1
2R4L4L5L4L5L-2L5R4R1L3R0L	638,1
3R1R2L3R5L2L-1L5R3L4L0R2R	608,6

Die beste evolvierte Kombination aus Bewegungs-FSM und Flag-FSM aus dem zweiten Experiment in Abschn. 6.2.2 ist für den Fall A in Tab. C.10 (Bewegungs-FSM) und Tab. C.11 (Flag-FSM) angegeben. Der Fitnesswert, den diese Kombination erreicht, ist $f = 168,4$.

Tab. C.12 zeigt spaltenweise die vier evolvierten FSMs, die alle Ranking Sets aus dem Experiment, das in Abschn. 6.3.1 beschrieben wird, komplett erfolgreich lösen konnten. Aus Platzgründen sind die Schrägstriche in den Genomen weggelassen worden.

Tab. C.13 zeigt die nach Anpassung beste FSM für den Fall mit 256 Agenten des Experiments aus Abschn. 6.3.2. Die nach Anpassung optimale Zufallswahrscheinlichkeit liegt bei 1,8%.

Tab. C.4.: Das jeweils beste evolvierte System vom Typ XY_T aus jedem der 30 Durchläufe des GA für das Experiment aus Abschn. 6.2.1.

Genom (X und Y)	T	f
3R2R4L2R5L4L-3R0L1L5R0R3L 2L3L1R4L1R3L-2L0R5R4L1L3R	377	497,3
4R0L1R5L2R3R-4R2L1R5R3R2L 4L3L2R0L5R1R-5L3R1L0R2L4R	407	521,0
2R3R1L3R3R2R-5L0R1R2L3L4R 5R4R1R1L5R3R-1R2R5R5L1R3R	245	587,5
5L2L3R4R1L1L-2L4R3L5R0L3L 4L3L4L3L4L4L-3R0L5L2L1R4R	187	558,1
1L4R0R0L3R1R-2L4R0R5R3R2L 4R1R3R5L0R5L-3L4L0L5R2R1R	418	574,0
2R2R5L4R4R4R-1L3R5R4L2L0R 4R2R4R1R5R1L-1R3R1R4R5R4L	245	587,5
1R2L4R2R5L1L-3L2L1R5R0L1R 4L3R2R3R5L2R-5R2L3L4R0L1R	16	593,4
4R2L4R1L3L3R-5L2L1R5R0L2R 0L2R2R2R5L1L-5R0L3R1R2L4L	106	567,2
2R3R2R0L4L0L-2L3R1L5L0R4R 1L2L4R2L5R1R-1L5L3R0R5L1R	257	569,4
1R5R0L0L3R2R-1R4R5L0R2L2R 5L0L2R4R1L3R-3L0R4L5R1R2L	364	560,2
3L3L5L3L3L5L-5R4L0L2R3R1L 5L4L1R4R3R1R-2R5L1L2L1L1R	16	579,5
3L4L1L2R3R2R-3L0R5L0R1L4L 1R1R5L4R2L2L-1L3L0R5R2R4L	379	580,2
3R4L5R2R1R0L-3R4R5R1R2L2L 5L2L4L2L3R3R-5L2R4L0R3L1R	392	518,9
1R2R5L5R2R0L-4R0R5R5R2R3L 2R1L1L1L5L5L-4L5R0R2L1R3L	301	573,7
1R2R4R5R0L3L-1R5R4R2L2L3R 4R4R4L4R0R1L-5L3R0L4L2R1R	387	526,9

Genom (X und Y)	T	f
3R4L1R1L5L3R-2L2R5R0L5L4R 3R0L2L4R3R3R-1L5R3L4L0R2R	146	587,4
1R0L5R0L2L3R-1R4R4R4L2L2L 0L4R2R1R5L5L-2L0R4L1L5R3R	92	553,9
3R1L5R3R0L0L-3L0R5R2L1R4L 5L4R1L5R3R1L-5L2R0R5R3L3L	257	569,4
3R2R2R3R2R0R-3R2L0L5L1R4R 3L0L1R0L3R0R-3R4L3R2L5L2R	261	568,5
5L5L1R3R5R5L-1L5R3L4R0L2R 3R4R1R1L5L2L-2L3R1L1L0L4L	300	586,2
5R2R0R1L5L1L-5R3R0L4R2R0L 4L1R2R2R3L1R-3R4R1L5R0L2L	271	578,9
2R0L2R3L0L1R-2L4L5L0R3R1R 5R5R1L1L2L4R-2R3R0L4R5R0L	112	577,1
2L1L2L1L2L2L-1R5L3R0L2R4L 2L3R4R1R3L4R-5R5L4R4L2L4L	16	579,5
4L3R3L0R3R5L-2L3L4R5R1R0L 4L2L5L1R1R4R-4L3L0R5L0R2L	251	580,1
3R4R3R5L4R4R-2L4L1R5R3L0R 5R0R0R1L1R3R-1R4R3L5L5R3R	245	587,5
5R0L3R2L1R4R-5R4L4L2R1R3R 1L4L0L1L4L2R-2L5R4R1R3L0L	395	534,5
2R3R2R0L0L5L-2L3R1L4L5R0R 3L3R0L5R1L2L-3L3L5L0R1L4L	215	573,8
4R4L3L4R1L1R-5R4L3L2R3R1L 3R4L3L0R3R3R-1L4R0L2R5R3L	18	597,8
5L3L5R0L0R1R-5L4L3L0L2L0R 4R5L5L2R0R5L-3R0L5R4L2L1R	391	557,7
5L1R3R1L2L4R-3L4L0R5R2R1L 4L3L1R5R3R0L-1R3L4L1R0L2L	264	566,6

Tab. C.5.: Das jeweils beste gemeinsam evolvierte System vom Typ UV_T-u aus jedem der 30 Durchläufe des GA für das Experiment aus Abschn. 6.2.1.

Genom (U und V)	T	f
3L0R0R5R3R1L-2L4L0R0R5R3L 0R4R3L5R2L4L-1R0L4L4R1R4L	18	411,4
3R1R0R0L5R5R-3R4L3L0L0L4R 2R4L3R3R3R3R-1R5L4L5R5L3L	18	415
5R2R2R4R4R3L-2R2L1R4L4R1L 3R5L4L1L0R2R-3R3R5L0L2R0R	22	423,9
3L1L0L1L4R0R-5L4L0R2L1R1L 4R3R5R1L0R0R-2R3R1R1L2L1L	18	443,6
1R2R1L5R3R2L-4R3R5R5R3L3L 1R0R0R5L3L5L-1R4R1L2R1L1R	48	405,2
1R0R0R3R3R3R-1L2L4L5R5L3L 3R3L0R0L2R1L-3R2L2R0L5R4L	18	445,6
3L5L0R0R3R1R-3L5L2R0R1R2L 4R5R1R2L0R3L-5R2R4L0L2R4L	18	420,5
5R5L1R4R0R5R-5L0L1R4L5R0R 3R4L5L2R1R3L-3L4L5L4L1R0L	18	455,8
0R5L0R4L2R3L-1R0L3R4L5R0L 4R3R1R1R5L3R-2R4L1L5L5L4R	19	449,5
4L4L3L1L1R5R-4L4L0L5R0R2R 3R4L3R0R2L1R-5R5R0R5R1L3L	18	440,4
3R3L0R1R0R4L-4L3L5L1R1R2L 2R2R5R3R4R3L-5R4L1R5L1R4L	18	432,2
4R2R4R1L0R3L-5R0L5L5L0R0L 2L2R4L5R0R3L-5L0L2R5R1R3L	19	469,2
3L3R0R0R2L0L-3L3L0R0R5R4R 2R4R0R1L5R3L-5R2L1R0L1R2L	18	420,5
2R2R5L5R2L2R-3R2R5L5R5L2R 0L3L1R5R5L0L-1L0R3R1L2L4L	18	398
5L1L0L2L1R3R-2L2L0R4R1R1R 5R4L3L5R1R0L-2R4L4L1L1R3L	18	427,8

Genom (U und V)	T	f
5L2R1L4R4R3R-1L2R1L3L1R2R 5L0R1L0R3R2L-4R5L0L4R3L4L	18	424,2
2R4R3L5L2L3R-5L5L1R5L5L3R 4R3L2R3L1L5L-4R5L5R5L1R2L	19	445,6
5L5R3R3L0R0R-5L5R4L3R2R0R 5L1L4L3R0R2L-2L3L4L1R1L3L	19	450,8
2L3L0R5L4R2R-2L2L0R4L4R2R 4L2L2L2R0L1R-3L2R1L4L5R1L	18	446,2
1R3R5R2L4R3L-1L4L4L5R2R2R 3L0L0R0R3R5R-3L4L1L0R2L4R	18	444,4
3R2R2L1R4R0L-4L0L5R4L3R3R 5L5R4L0R5R4L-3L2L3R0R3R4L	41	450,2
2R1L0L4L3R0L-4L3R5R4L3R4L 0R5R1R4L3L2R-5R3L5R4R2L0L	18	412,4
1L0R3L5R1L4R-1L0R1R0R0L4L 5R2L3L4R1R0R-1R5L0L5L3R3R	18	420,5
1R1L4R2L1R5R-3L5L2L5L0L1R 5L4R3R2L3L4L-3R4R1R4R1L5R	37	445,5
4R5R2R2R0R1R-5R3R5R2L2L0L 5L1R4L2L0L0R-5L0R3L4R2L0R	18	447,8
0L3R4R1R3R2L-1L0R5L2R1R0R 3L4L4R5L0L1L-3L3R4R2R2L4L	41	442,4
1L0L0R4L3R0R-5L0R1R4L0R0R 5R3L2L4R3L0R-1L3L1R1R0L2L	40	409
2R3R5L5R1L2R-3R2R5L5R0L2R 0L2L0L2L2R4L-5L4R3L1L5L0R	18	398
4L0R1L2R0R1L-4L5L2L1R0R3R 1R0R5R4L4R2R-3R5R5R1L0L1L	18	423,8
4L1L5R2L0R3R-4L3R0R5R0R1R 3L1R2L4L5L0R-4L2R1L1L5L2L	19	450,8

Tab. C.6.: Das jeweils beste gemeinsam evolvierte System vom Typ UV_T-m aus jedem der 30 Durchläufe des GA für das Experiment aus Abschn. 6.2.1.

Genom (U und V)	T	f
2R4R4R1L1L0L-5R3L0L1R5L0L 3L2L1L5L2L2R-4R3R4L1L2R0R	40	411,8
2R5R0R4L3L1R-4R2R5L4L0L4R 3L4R5R1R2R1R-3L4R4R0R0L3L	40	407,6
1L5L0L1R4L3R-4R0L4L2L0L0L 3R2L0R4L3R0L-3R2L1R1L2R1L	44	464,4
2R5L5L3R4R2L-1R0L3L1R3L4R 3L4L5L0R2L0L-3L0R4R0R0L3R	38	440
4R1R2L1R1L3L-5R2R1L1R0R2L 1L4R0R4L3R2L-1L0R3R0L5R0L	41	442,2
1R0R5R2R0L1R-4R3L4R4R0L4L 2L4L4R4R1R1R-2L0L0R0R3L5R	38	426
2R0R0L0L0L0R-2R2L0L5L1L1R 4L1L3R2R2L0R-5L5L4L0L1R0R	38	424,8
1R3R0R4L3R0L-5R0L1L5L2L0L 3L4L0L1L0L4R-2L2L0R2R4R5R	42	432,8
5R1R0R2L2L3L-5R5L0R2L2R0L 1L1R2R2R5R5R-3R5L0R4L5L4R	46	427,4
3R2R2L2L4R5L-4R0L5L1L5R4L 5L4R0R5R2L1L-2L2R0R2R3L3L	43	456,8
5L4R3R3R5L0L-1R4R4L2L1L0L 1R4R0R4L2R0R-3R5L0L0L2R3R	42	411,6
3R0R4R5L2L4L-3R5R4R0L0R4L 5L0L1R0R2L4L-5L5L1R4R3L4R	44	380,1
1L0R3R3L5L4R-1L0R4R0L5R1L 5R2L3R5R0L2L-4R4R5L2R0L1L	40	422,2
2R5R3R4L3L2R-4R4R4R4L1L0L 3R5L5R1R4R2L-4R3L3R1R1L1R	46	402,2
3R2R4R0L5R3L-3R1L1L0L3L2L 5L2R3L4R3R4L-3L3L1L0R3R1R	42	410,4

Genom (U und V)	T	f
5R1R3L3R3R0L-5R3R0R3R3R0L 5L4R4R2L3L4R-2L2L0R1R2R1R	48	411
1R0L0L5R3L4R-1R0L2R4L4L3L 0L5R5L4L5L4L-4R3R1L1L2R4L	38	397,2
1L4L4R0L2R1L-2L3R0R2L2R1L 4R0R3L5R0L1R-4R5L0L1L0L0L	38	437,4
5R4L1R4L4L0L-5R2L1L3R5R0L 2L4L3L1L0L2R-5L3L5L0R3R2R	38	411,4
4R5L5L4R3L0L-2R0R3L4R3L1R 3L3L5L4L5R4R-1R3R3L1L0R2L	38	436,2
4L1R2R1R3L3R-5L0L0R2L0L0R 1R3L4R2L2L2L-1R0L0R1R2L0L	48	414,2
2L2L5L0R1R4L-3L4L5L0R0L3R 0R5L3R4R3R1L-5L2R4L4R5R4L	42	377,6
4L5R1R2R0L1R-3R4R4R1L1L2L 2L0R5L5L1R3R-1L0R0R4L4L1R	42	382,6
4L5L5L0R0R2L-4L5R1R3L0R4L 0R2L3R2R5R0L-1R4L3R4L1R0L	38	384,8
1R0L5L4R1L0R-1R2L5L0L4L2R 5R2R0L4L1R4L-2R3L0L2R2L3L	44	394,6
4R4R0L5R0L1R-4R4L2L4R0L2L 1R5L4R5L2R2L-4L4R4R4L3R0R	42	412,2
4R5L5L2R3L4L-3L3L3R1R3R0R 5R3L2L0L5R0L-5R4R3R4L4L0L	44	438,8
2R4R5L4L3L2R-1R4L5L4R2R2R 1L3L2R2R0R3L-4L5R3R1L0R4L	38	395,2
5L3L4R2L1R0L-3R4L4R1R1R1L 5R2L3R2R0R3R-1R2L1R4L1R3R	40	375
4R2L5L5L0L0R-5L4L5L4L2L2R 5R4L3L4L5R2R-1L4L1R0L1R5L	40	425,9

Tab. C.7.: Das jeweils beste evolvierte System vom Typ $U_T V_T$-u aus jedem der 30 Durchläufe des GA für das Experiment aus Abschn. 6.2.1.

Genom (U und V)	T_U/T_V	f
2R1L0L4R4R1R-2R5R0L1R2R3L 5R2L5L2L4R3R-4L4L0L2R1R4L	30 4	385,9
3R1R2L4L2R1R-5R2R1L3L0R1R 5L4L3R0L3R4R-5L2R1L4L3R4R	35 41	456,8
5R5R3L4L1L3L-5R2R3L4L0L3R 4R4L2R4L0L1L-4R4L5L4L0L1R	4 37	382,2
2L5R3L5R4L1L-1L5R0L2R2R1L 4L5R3R5L2L3L-2L5R1L5L3R1L	60 12	369,4
2R3L3R1L3R1L-1L5L5R2R5R2L 1L5L5R0R5L5R-3L0R4R0R5L4R	36 3	462,4
5L1L0L1L3L4L-2R5R0L3L4L0L 4L3R0R0R0R1R-4L3L2R1L0R5R	2 36	381,8
1R1L0L4L2R3R-5R3L5R1R3L0L 4L3R4R5L0R3L-4L2L3R3R0R0L	10 30	385,6
2L3R4R2L0R1L-3L4L5R2L1L0L 3R2L3R0R2L0L-1R0L1L0R1R2L	1 35	440,8
2R4R0L5R0R2R-2R0L0L4L4L2L 1L3L2L1R2L0R-5L2R4L4L0R0R	48 28	373,8
2R0L0R1L5L0L-2R3L3R2L5L4L 4L5R2L3L2R0L-5R5R2L4R3L1L	30 6	444,1
3R3R0L0L5L0R-3R0R1L0L0R0R 1R1R4L5R0R2L-3L0R0R0R1L5L	30 6	364,2
3L4R1L2L4R0R-5R3R3L0R4R0L 2L3R0L1L1R5L-1L3R4L1L5R0L	2 34	366,6
1L1L4R5L3R4L-2L5L4R0R2L3L 4R5R3R5L0L1R-4R0L4R1R0L4L	4 30	377,4
1R0L5L5R5R0R-1R0L2R0L2R2L 4R3L1R5L4R0R-2L1L0R0R0R4L	30 6	364,2
2R4L4L5L5L1R-1R4L0L4R3L3L 1R0L3R5L5R3R-1R0L4L2R2R3R	10 28	385,5

Genom (U und V)	T_U/T_V	f
3L4L3R0R1R4L-4L4L5L2L1R0L 4L2L0R3L3L0R-2R4L0L5L0R4L	34 2	363,2
4L1L5L5L1R2L-5R0L5L0R3R0L 1R4L5R0R2L1R-2R4L4R4R2L2L	2 40	395,4
2R5L0L1L5L3R-2R0R0L0R4R4L 2R4R4R0L3L1R-5L5L2L4L0R1R	32 6	368,6
4L5L2R5R4R4R-3L4L0R0R1R2R 2R4L1L0L1R5L-3R3R0L0L0R0R	2 28	373
3R2L5R2R5R1R-3R4L2R0L1R3R 1L2R4R4R0L1R-4R2R4R5R2L4L	4 29	444,1
0L2R1R2R3L4L-5L3L3L4R5L0R 5R3R1R1L3R5L-3R3R3R1L0L0L	14 24	439,4
4L5R0L1L0L0L-4L3R4R4R3L2L 4R5L3L4R3L0R-5L5L0L4L1L1R	10 32	350,8
2R3R1R1L0L4R-3R3R1R5L0L4R 2L2R3R4R0L2R-2L2L0R2L0L4R	2 30	340,1
2R2R0L4R4R4R-2R1L0L4R0L3R 3R1R0L2R1R4L-2L5R0R5L1L2L	30 8	400,2
4L2R0L1R0R5R-2R2R1L5R1R5R 1R3R3R5L2R3L-4L0L4L2R0R4L	34 8	357,2
5L3L4L0R4R0R-5L1R1R1L0R0R 3L5R4R5R1L2R-3L4R4R2L1L2R	34 4	342,6
3L3R4R0R5L5L-3L3L4R0R2L4L 3R1R4R0R2R1L-2R2R1L5L2R4L	58 28	401
1L4R3R5L4L5L-5R2L1R5R1L3L 1R3R1R4L3R0L-4R3L0R4L3R2L	6 28	378,4
3L2R4R3L1L2R-5R2R5R0L5R0L 2L5L1R5R5R4L-2L2L1R3R2R1R	2 32	370,6
5R5L4L0R2R3R-2R3R4L3L2R4R 5R3R0L2L1R3R-4L4L3L1R1R5L	32 6	359,2

Tab. C.8.: Das jeweils beste evolvierte System vom Typ $U_T V_T$-m aus jedem der 30 Durchläufe des GA für das Experiment aus Abschn. 6.2.1.

Genom (U und V)	T_U/T_V	f
4R4R1R4R1R0L-5R5L0R2R3L0L 0R0R5L2R3R0L-1R0L3L2R2L4R	69 38	445,2
3L4R5R4L3L3L-2L0L0R4R1L3L 0L1L2R4L3L4L-2L1R0R5R4L2L	81 33	422,8
2L0L5R3R5L1L-3L3R4L0R0R3L 5R5L3L0R4R0R-5L3L3R1R4R2R	35 60	425,8
1R2L5R3L3L3L-2R1R4L4L3R5L 3L4L0R0R5R3R-3L4R0R0R4L0L	26 50	445,8
2L5R0L4L0R5R-4R4R5R1L1L3R 3R4L5R1L3L2L-2R2L5R1L1L2L	24 54	423,6
2L5R0R0R2R0L-5R5R1R4R3R1L 5R3L1R1L0L2L-4R2L1R1L1R3R	60 36	356,9
1L5L0R0R5R1R-2R5L5L0L1L1R 4R2R2R0L1R1L-5R2L1R2L2R3R	54 26	415
5L3L4R3L3L1R-4L3R1L1L2L2R 1R0L4R4L5L0R-1R0L4L2L5R2R	20 60	419,1
3L5R5L1R5L5L-2L0R4R4R2L0L 3R3L3R4R3L2L-3R1L5R4R5L2L	39 73	439,6
4R5R5R3L0L0R-4R4L2L2L0L4L 0L2R5R1R3L4R-4L3R4R4L0R0L	50 24	427,8
2R2L4L3L5L1R-3R5L0R0L0L0R 5L4R0R5L1L1R-2L4L0R4L0L3R	39 55	442,4
1L5R0R1R0L4R-1L5R5R0R5L4R 3L5R4L2L1L4L-2R3R3L1L0L1L	76 10	400,2
1R3L2L3R4R0R-4R5R4L1L2R2L 1R4R3L2R4L1L-2R0R3L2R5L5R	24 53	434,2
4R5L1L3L4L5R-3R2R1L5L2L3R 3R5L1R5R5L3R-1R0L0R5L0R4R	40 85	435,2
5L5R2R4L2L0R-4R2L1R1L1L2R 1R2R3L0L5L3R-4R2R4L2L0L3R	40 60	440,2

Genom (U und V)	T_U/T_V	f
3R1L1R4L4R0R-2R4L1L5L1R5L 2L2R0L2R4R0L-2R5L1L5R2L3L	35 63	428
0L3R4R5L5R1L-2L1R0R2R0R2L 1R3R0L4R3L3R-2R3R0L2L0R0R	37 60	452
0L0L0R1L0R4L-1L0R5L3R5L2R 5L3R3L2R1L0R-4L4R4L0R1L0L	42 54	439,2
2L1L2R5R5L5R-3R5L4R5L2L1R 1R5R4L3R1R1L-2L5R4L5R2R2L	44 63	445
5L4L2L3R4R4R-3R3L3L2R5R4L 4L0R5L2R2L2L-1L0R1R2L2L0L	41 60	426,2
1R5R2L0L3L1L-3R0L0R0L0L3L 4R2L2L5R3L5L-1L2R1L4L1R1R	56 37	418,4
1L4R4L5L1R3R-2R3L3R5L4L3R 4R2L1L3R0R3R-1R0L5R1R3L0L	46 30	365,2
1L4R3L2L5L5L-5R0L0R2L2R0L 1R0R0L4R1R0L-5R0L4L4L2R4R	40 67	455,4
1L2R4L3L0L5R-5R2L1R5L0L3R 2R0R3L5L0R3L-2R3R0L5R5R0R	35 71	453,6
0R5L0R1L3R4R-5R3L5R0L5R0L 5R2R1L1R3R3L-1R2R1L0R1R4L	24 52	422,8
4R2R3L0R2L0R-3L2L3L2R2L0R 5R4L1L2L1L0L-1R3L5R1R1R1L	40 8	413,2
2L5L5L2R2L3R-4L5R1L0L0R4L 3R5R3L0L2L1L-3R2R0L0L0L2R	26 48	416,9
2R0L3R1L2R5L-1R0L3R1L3R3R 4L3R2R3L1L2R-5R2R3R2L5L1L	81 88	440,2
5L1L0R3R5R4L-3L3L1L1R2L4R 2R4L3L4L1R3L-2R5L0L4L0R4L	36 50	425,8
0L4L0R5L3L2L-1L0R4R3R0L1L 2R2L1R5L0L2L-4R4L1R4R0L5R	39 62	443,8

Tab. C.9.: Die für jede Entscheidungsmenge in 30 GA-Durchläufen beste evolvierte FSM aus dem Experiment in Abschn. 6.2.2.

Typ	Genom	f
U	2/Rx 2/Rx 3/Lx 0/Lx 3/Lx 1/Lx - 2/Lx 4/Rx 5/Rx 0/Lx 3/Rx 1/Lx	605,6
V	5/Lx 0/Rx 4/Lx 1/Rx 5/Sx 1/Rx - 5/Lx 0/Rx 0/Sx 2/Lx 3/Sx 2/Rx	470,6
W_0	3/R0 4/L0 2/S1 0/Sx 1/L0 2/S1 - 4/R0 5/R0 3/S1 0/Sx 4/Sx 5/S1 3/S1 0/L0 4/R0 2/R0 2/Sx 1/Sx - 2/R0 4/R0 1/Sx 0/R0 4/L0 1/R0	330,1
W_1	5/S0 0/R0 4/Sx 2/R0 0/L1 1/L1 - 3/S0 0/S0 4/L1 3/S0 3/L1 2/L1 1/R0 3/R0 3/L1 4/S0 4/R0 5/L1 - 0/S0 5/Sx 2/Sx 2/R0 2/R0 5/Sx	376,5
W_2	4/R0 4/L1 4/S1 1/Sx 2/R0 0/Sx - 0/Sx 4/R0 2/L1 4/S1 4/S1 0/S1 4/R0 4/L1 5/L1 4/L1 5/L1 3/S1 - 0/R0 3/R0 0/L1 3/S1 2/L1 5/S1	344,8
W_3	5/S0 1/Sx 2/R0 2/Lx 5/R0 3/R0 - 3/R0 0/Lx 4/R0 2/R0 4/S0 5/Sx 5/Sx 1/S0 3/S0 1/R0 5/R0 4/Lx - 5/Sx 5/Sx 0/S0 4/R0 4/R0 0/Sx	325,5
W_4	0/S1 2/S1 3/Sx 4/Lx 5/R0 5/Lx - 4/S1 3/Lx 2/S1 2/Lx 4/Sx 3/Lx 1/R0 2/R0 5/S1 4/Lx 5/R0 1/Sx - 5/S1 5/Lx 5/Lx 5/Lx 4/R0 4/Lx	319,6
W_5	2/L1 5/Sx 4/S0 4/Sx 5/L1 2/R1 - 3/L1 0/L1 2/L1 4/L1 5/L1 5/S0 2/S0 3/R1 2/R1 1/R1 4/S0 2/R1 - 3/S0 5/S0 4/L1 3/Sx 1/L1 1/S0	378,6
W_6	5/Lx 4/S1 5/Lx 1/R1 4/Lx 3/Lx - 0/S0 1/S0 5/Lx 1/R1 2/Lx 5/S1 3/Lx 5/S1 0/S1 0/S0 5/R1 1/R1 - 1/R1 4/S0 4/S1 4/S0 1/S0 4/S0	340,2
W_7	0/Lx 3/R1 3/Lx 0/Lx 0/R1 2/S0 - 1/R1 3/R1 4/Sx 3/S0 5/R1 1/R1 0/Lx 3/Lx 1/R1 1/Lx 1/Sx 3/S0 - 3/S0 5/S0 5/Sx 0/S0 5/S0 5/Sx	376,8
W_8	1/Rx 0/Rx 5/Lx 5/Lx 4/Lx 3/Lx - 3/Rx 2/S0 3/S0 2/S0 0/Rx 4/Rx 3/Rx 1/S1 3/Lx 2/Lx 0/S0 3/Rx - 4/S1 3/Rx 5/Rx 3/S0 4/S1 4/Rx	332,4

Tab. C.10.: Bewegungs-FSM für das beste System A aus Abschn. 6.2.2.

Inputs			aktueller Zustand							
F_f	ack	m	0	1	2	3	4	5	6	7
0	0	0	4/Rg	4/Sr	5/Sr	0/Bg	1/Bg	5/Lr	1/Lg	3/Sg
0	0	1	1/Lr	4/Rr	3/Rg	5/Br	4/Lg	1/Sr	0/Sg	3/Rr
0	1	0	1/Lr	0/Sr	5/Lr	4/Bg	3/Sg	2/Rr	4/Lr	5/Bg
0	1	1	5/Sg	4/Br	2/Sr	1/Lg	0/Sg	0/Lg	3/Bg	6/Rr
1	0	0	4/Lg	4/Sr	3/Sg	1/Lg	1/Rg	7/Sg	7/Sg	1/Lr
1	0	1	0/Bg	1/Sg	4/Rg	4/Br	0/Rg	7/Lg	7/Sg	1/Rg
1	1	0	0/Br	1/Sr	1/Rr	6/Bg	0/Sg	2/Rg	0/Lg	2/Br
1	1	1	7/Rr	3/Bg	3/Sr	5/Rr	3/Br	6/Bg	1/Lr	1/Bg

Tab. C.11.: Flag-FSM für das beste System A aus Abschn. 6.2.2.

Inputs		aktueller Zustand							
F	ack	0	1	2	3	4	5	6	7
0	0	0/C1	1/C1	7/C0	3/C1	6/C1	1/C1	3/C1	6/C1
0	1	3/C0	4/C0	2/C0	2/C0	0/C1	5/C0	6/C1	2/C0
1	0	6/C1	6/C0	3/C0	3/C1	7/C1	4/C1	3/C1	1/C1
1	1	6/C0	4/C1	3/C0	6/C1	1/C1	4/C0	3/C0	6/C0

Tab. C.12.: Die vier komplett erfolgreichen, evolvierten FSMs für das ARP-Experiment in Abschn. 6.3.1.

Input	FSM 1	FSM 2	FSM 3	FSM 4
0	2L1B1B3B3R3L	2L0B0L3L3R5L	2L4B3L3B1R1R	2B2B4L4R3S2R
1	5S4R4R3R3S3L	5S5R4R3R3S3L	5S1R4R3R3S3L	5S4S4R3R3S3L
2	4S3L0S5S3S1S	4S3L0S5S3S1S	4S3L0S1S5S3S	4S4L0S5S3S1S
3	3S1B3L2L0L0B	3S1R3S2L0L0S	3S0S1L2L0L5R	3S0B4S2L0L3S
4	0B0L0B2R3R4R	4B2R2L2R4R4R	0B1B0R2R4R4R	0R4B5R2R2R4R
5	3B2S4S1L2B4R	4B2B5S0R0B0B	4B3R5B1L2B3R	0B3B2L1L2L0B
6	4B0R5R4R5R2S	4B1B0B4R1R3L	4B2S4L4R4R5R	4B0L4B4R0L3L

Tab. C.13.: Die beste randomisierte FSM für den Fall mit 256 Agenten des ARP-Experiments aus Abschn. 6.3.2.

Inputs		aktueller Zustand							
m	Zone	0	1	2	3	4	5	6	7
0	0	1/R	6/S	2/S	3/R	3/R	3/L	6/S	4/S
0	1	5/R	3/R	5/S	0/L	0/R	7/R	6/R	7/R
0	2	0/R	2/R	5/R	4/B	6/B	5/B	6/B	7/R
0	3	6/S	0/S	4/S	4/S	4/S	7/S	4/S	1/S
0	4	3/R	4/L	1/L	1/S	5/S	2/S	7/S	5/S
0	5	6/B	7/B	0/B	5/L	2/S	1/R	1/R	4/L
0	6	2/S	1/S	6/L	1/S	4/L	7/S	6/L	2/S
0	7	3/L	7/L	1/L	0/L	7/S	2/L	2/L	4/L
0	8	7/L	4/L	2/L	4/L	6/B	0/L	5/B	3/L
1	0	4/S	1/R	3/R	5/R	3/R	4/S	4/S	4/S
1	1	1/S	5/R	0/R	2/R	5/R	0/R	2/R	7/R
1	2	0/R	2/R	6/R	6/R	2/R	5/R	1/R	5/R
1	3	2/S	5/S	3/R	0/L	2/S	4/S	6/S	3/S
1	4	3/R	6/S	7/L	2/S	6/S	0/S	0/S	5/S
1	5	7/L	1/L	0/L	5/L	3/R	2/B	4/B	1/R
1	6	1/L	7/S	6/S	7/S	6/S	6/S	4/L	6/L
1	7	3/L	7/L	6/L	3/L	0/L	4/L	5/L	1/L
1	8	3/L	5/L	5/L	7/L	4/L	5/B	0/L	3/L

D Glossar

- *Agent (im CA-Modell)*: Ein (aktives) *Objekt*, welches autonom, perzeptiv und reaktiv ist. Der Agent besteht aus *Sensor*, *Aktuator*, *Kontrollfunktion*, sowie *Datenzustand* (vgl. Abb. 2.1).
- *Agentenarchitektur*: Die Einheit aus *Sensor*, *Aktuator* und falls vorhanden *Aktionsarbiter* mit Schnittstelle zur *Kontrollfunktion* auf der einen und zur *Umwelt* und *Datenzustand* auf der anderen Seite (Abschn. 3.2).
- *Agentenprogramm*: Konkrete Implementierung, mit der die *Kontrollfunktion* programmiert wurde, also die inhaltliche Füllung der strukturellen *Kontrollfunktion* (Abschn. 2.1.2).
- *Agentenwelt*: Die Gesamtheit aller *Agenten* und anderer Elemente im MAS (vgl. Abb. 2.2).
- *Aktion*: Die Ausgabe des *Aktuators*, die Veränderungen des *Datenzustands* und des Zustands der *Umwelt* induziert (vgl. Abb. 2.1 und Abb. 3.4).
- *Aktionsarbiter*: Erhält als Eingabe die *Aktionen* (möglicherweise von mehreren Agenten) und löst Konflikte sich widersprechender Aktionen auf. Die Ausgabe sind (elementare) *Wirkungen*, die als *Basisoperationen* den *Rechenfunktionen* zugeführt werden (vgl. Abb. 3.4).
- *Aktionsmapping*: Eine Abbildungsfunktion im *Aktuator*, die Entscheidungen und weitere Inputs auf Aktionen abbildet (Abschn. 3.1.2).
- *Aktionsmenge*: Die Menge aller möglichen *Aktionen* eines bestimmten Agenten.
- *Aktuator*: Funktion, die die *Entscheidung* des Agenten möglicherweise unter Berücksichtigung von weiteren Bedingungen auf *Aktionen* abbildet (vgl. Abb. 2.1 und Abb. 3.4).
- *Attribut (eines Objekts im MAS)*: Ein zum Objekt gehöriger Zustand, der intern (also für das Objekt bzw. seine Verhaltensregel) und/oder extern (für andere Objekte auf der Zelle oder in Nachbarzellen) sichtbar ist. Alle Attribute bilden gemeinsam mit dem *Zelltyp* den Zustand der Zelle. Attribute von Agenten sind ein Teil ihres *Datenzustands* (vgl. Abb. 3.1).
- *Ausgangskonfiguration* oder *initiale Konfiguration*: Die Konfiguration eines Zellularen Feldes zum Zeitpunkt 0.
- *Basisoperation*: Ausgaben des *Aktionsarbiters*, die in der *Rechenfunktion* die Veränderung eines einzelnen *Attributs* oder des *Zelltyps* bewirkt (vgl. Abb. 3.4).
- *Datenzustand (eines Agenten)*: Die Gesamtheit aller *Attribute* eines Agenten. Die *Aktion* kann auf den eigenen Datenzustand ebenso eine *Wirkung* haben, wie auf den Zustand der Umwelt (vgl. Abb. 2.1).
- *Entscheidung*: Die Ausgabe der *Kontrollfunktion*, die vom *Aktuator* in eine Aktion umgesetzt wird. Nicht zu verwechseln mit der *Aktion* (vgl. Abb. 2.1 und Abb. 3.4).

- *Entscheidungsmenge*: Die Menge aller möglichen *Entscheidungen* eines bestimmten Agenten.
- *Fitness*: Die Fitness eines Kandidaten in einem Genetischen Algorithmus ist seine Qualitätsbewertung. Über den Fitnesswert können mehrere Kandidaten miteinander verglichen werden.
- *Fitnessfunktion*: Die Abbildungsfunktion der Kandidaten in einem Genetischen Algorithmus auf seine jeweilige Fitness.
- *Folgezustand*: Der Zustand einer Zelle im nächsten Taktschritt.
- *Frontzelle*: Die *Nachbarzelle* eines *gerichteten Agenten* in der Richtung des Agenten. Nicht zu verwechseln mit der *Zielzelle*.
- *Gen*: Unteilbarer Bestandteil eines *Genoms*. Hier bestehend aus einem Paar *Folgezustand* und *Entscheidung* (Abschn. 5.3.2).
- *Generation*: Die *Konfiguration* des gesamten Zellularen Feldes zu einem bestimmten Zeitpunkt.
- *Genom*: Repräsentation eines Individuums (eines potentiellen Lösungskandidaten) in einem Genetischen Algorithmus. Hier die Repräsentation einer FSM als String (Abschn. 5.3.2).
- *Inputreduktion*: Funktion, die als Bestandteil des *Sensors* oder dem Sensor vorgelagert, die Inputs der Nachbarzellen auf weniger Inputs für die *Kontrollfunktion* reduziert (Abschn. 3.1.2).
- *Kollision*: Tatsächlicher, nicht erlaubter Zusammenstoß mehrerer Agenten auf einer Zelle. Nicht zu verwechseln mit *Konflikt*.
- *Konfiguration*: Die Gesamtheit aller Zustände in einem Zellularen Feld.
- *Konflikt*: Situation, in der sich mehrere Agenten auf eine Zelle bewegen würden, und damit eine nicht erlaubte *Kollision* verursachen würden.
- *Konfliktzelle*: Die Zelle, in der ein *Konflikt* auftritt.
- *Kontrolleinheit*: Die Gesamtheit aus *Sensor*, *Kontrollfunktion* und *Aktuator*, also der Teil des Agenten, der die Zustände der Umwelt auf eine Aktion abbildet. Nicht zu verwechseln mit der *Kontrollfunktion*.
- *Kontrollfunktion*: Die Abbildung von *Wahrnehmung* auf *Entscheidung*. Nicht zu verwechseln mit der *Kontrolleinheit* oder dem *Agentenprogramm* (vgl. Abb. 2.1).
- *Kontrollzustand*: Zustand innerhalb der *Kontrollfunktion*, von dem die Entscheidungen abhängen. Der Kontrollzustand ist kein Teil des *Datenzustands*, kann aber in den Datenzustand ausgelagert werden, wenn der *Sensor* und der *Aktuator* entsprechend angepasst werden.
- *Mediatorzelle*: Eine Zelle, die gleichzeitig *Frontzelle* von mehreren Agenten ist, die in dieser Situation kommunizieren können (All-to-All Communication Task).

- *Multiagentensystem (MAS)*: Ein (lose verbundenes) Netzwerk von *Agenten*, die sich in einer *Umwelt* befinden und ein bestimmtes *Verhalten* haben.
- *Nachbarschaft*: Die Menge aller Zellen, deren Zustand ganz oder teilweise von einer Zelle zur Berechnung der Zellregel gelesen werden kann. Die eigene Zelle kann eingeschlossen sein.
- *Nachkommenschaft*: siehe *Offspring*.
- *Objekt (im MAS)*: Eine zusammenhängende, in sich abgeschlossene Einheit im MAS, die als Ganzes immer in einer einzigen Zelle untergebracht ist. Agenten sind Spezialfälle von Objekten.
- *Objekttyp*: Gleiche *Objekte* (mit den gleichen Attributen und Regeln), werden unter einem Objekttyp zusammengefasst.
- *Offspring*: Eine Liste von Lösungskandidaten, die nach bestimmten Verfahren aus der *Population* gebildet wird.
- *Operation*: Die Gesamtheit aller *Basisoperationen*.
- *Population*: Eine Liste von Lösungskandidaten, mit der der Genetische Algorithmus aktuell arbeitet.
- *Rechenfunktion*: Funktion, die eine *Basisoperation* ausführt. Inputs sind eine von mehreren möglichen *Basisoperationen* und der Wert des *Attributs* oder *Zelltyps*, der aktualisiert wird (vgl. Abb. 3.4).
- *Rechenwerk*: Die *Rechenfunktion* inklusive des Zustands, den diese aktualisiert. Das Rechenwerk stellt einen Moore-Automaten dar .
- *Quellzelle*: Nur im Zusammenhang mit der *Zielzelle*, die Zelle von der aus sich ein Agent (oder ein anderes Objekt) wegbewegt.
- *Sensor*: Bildet den Zustand oder einen Teil des Zustands der *Umwelt* und des *Agenten* selbst auf die für den Agenten sichtbaren Teil (die *Wahrnehmung*) ab (vgl. Abb. 2.1 und Abb. 3.4).
- *Sichtbereich*: Die Menge aller Zellen, die für den Agenten zu diesem Zeitpunkt wahrnehmbare Elemente (für die Verarbeitung in seiner *Kontrolleinheit*) enthalten. Nicht zu verwechseln mit der *Nachbarschaft* der Zelle und der *Wahrnehmung*. Der Sichtbereich enthält die zu einem bestimmten Zeitpunkt wahrnehmbare Umwelt und den Agenten selbst.
- *Startposition*: Die Position im Zellularen Feld, an der ein Agent (oder ein anderes Objekt) sich in der initialen Konfiguration befindet.
- *Swap*: Ein Positionstausch zweier gerichteter Agenten, die sich in der *Frontzelle* des jeweils anderen befinden.
- *Transitionsfunktion*: Die Funktion, die den momentanen Zustand der Zelle und die Inputs (z. B. von Nachbarzellen oder globalen Parametern) auf den *Folgezustand* abbildet.

- *Umwelt*: Die Gesamtheit aller für einen Agenten zu irgendeinem Zeitpunkt sichtbaren oder manipulierbaren Elemente, exklusive des Agenten selbst. Nicht zu verwechseln mit dem *Sichtbereich* und der *Agentenwelt*. Der zu einem bestimmten Zeitpunkt wahrnehmbare Ausschnitt und der modifizierbare Ausschnitt der Umwelt sind zwei Teilemengen der Umwelt (vgl. Abb. 2.2).
- *Verhalten*: Die Gesamtheit der Abbildungen von *Wahrnehmung* auf *Aktionen* über die gesamte Laufzeit des MAS hinweg.
- *Wahrnehmung*: Der Teil der Zustände der Umwelt und des Agenten selbst, der durch den *Sensor* bestimmt wird und als Eingang in die *Kontrollfunktion* geht (vgl. Abb. 2.1 und Abb. 3.4).
- *Welt*: kurz für *Agentenwelt*.
- *Wirkung*: Die durch die Aktionen aller Agenten erzielte Veränderung eines Zustands oder Teilzustands in der Agentenwelt. Die Wirkung wird durch die *Operationsfunktionen* realisiert.
- *Zellregel*: siehe *Transitionsfunktion*.
- *Zellstruktur*: Die Gesamtheit aller strukturellen Elemente einer abgeschlossenen Zelle mit Schnittstellen zu anderen Zellen, beinhaltet also auch die komplette Zellregel.
- *Zelltyp*: Teil des gesamten Zellzustands, der angibt, welche *Objekte* sich gerade auf der Zelle befinden, und damit, welche der anderen Teilzustände (Attribute von Objekten) relevant sind (vgl. Abb. 3.1).
- *Zellulares Feld*: Die Gesamtheit aller Zellen, die zur *Agentenwelt* gehören, und deren Verbindungstopologie.
- *Zielposition*: Die Position im *Zellularen Feld*, die ein Agent in einem MAS erreichen soll (Agent Routing Problem). Nicht zu verwechseln mit der *Zielzelle*.
- *Zielzelle*: Die Zelle, auf die sich ein Agent (oder auch ein anderes Objekt) in diesem Takt tatsächlich bewegt. *Mögliche Zielzellen* sind solche, auf die sich ein Agent nach den Regeln des MAS (nicht nach dem Agentenprogramm) in diesem Takt bewegen darf. Nicht zu verwechseln mit der *Zielposition*.
- *Zustandsalphabet*: Die Menge der möglichen Zustände einer Zelle im CA. Zum Zustand einer Zelle gehören in einem MAS der *Zelltyp* und die *Attribute* aller *Objekte*.

E Abkürzungsverzeichnis

1P	One-Point Crossover
ANN	Artificial Neural Network (Künstliches Neuronales Netz)
ANTS	Autonomous Nano Technology Swarm
ARP	Agent Routing Problem
ATAC	All-to-All Communication Task
BDI	Belief Desire Intention
CA	Cellular Automata (Zellulare Automaten)
CAM	Cellular Automata Machine
CDL	Cellular Description Language
CEP	Creatures' Exploration Problem
CEPRA	Cellular Processing Array
DAI	Distributed Artificial Intelligence (Verteilte Künstliche Intelligenz)
DB	Double-Gene Mutation
EA	Evolutionary Algorithms (Evolutionäre Algorithmen)
EV	Every-Gene Mutation
FSM	Finite State Machine (Endlicher Zustandsautomat)
GA	Genetic Algorithms (Genetische Algorithmen)
GCA	Global Cellular Automata (Globale Zellulare Automaten)
GP	Genetic Programming (Genetische Programmierung)
IBM	International Business Machines Corporation
IGA	Incremental Genetic Algorithm (Inkrementeller Genetischer Algorithmus)
IRT	Input Reduction Table (Inputreduktionstabelle)
KL	Künstliches Leben (Artificial Life)
MAS	Multi Agent System (Multiagentensystem)
NASA	National Aeronautics and Space Administration
PAP	Particle Alignment Problem
PCA	Probabilistic Cellular Automata (Probabilistische Zellulare Automaten)
SG	Single-Gene Mutation
TS	Time-Shuffling Technique (Time-Shuffling-Technik)
UN	Uniform Crossover

F Symbolverzeichnis

Zellulare Automaten

A	Zellularer Automat
d	Dimension der CA-Koordinaten
g	Generation
l	Länge in einer Dimension
N	Nachbarschaftsbeziehung
r	Radius der Nachbarschaft
R	nicht-uniforme Menge von Regelmengen
S	Zustandsalphabet
t	diskreter Zeitpunkt im CA
z	Zelle im CA
γ	globaler Parameter im CA
Γ	Alphabet der golabelen Parameter im CA
ϕ	Zustandsübergangsfunktion
Φ	Zustandsübergangsfunktionsmenge
δ	nicht-uniformes Mapping

MAS-Modell im Zellularen Automaten und Beispielsysteme

a	Agentenaktion
a_b	bedingte Agentenaktion
b	Basisoperation
$\vec{c}$	Bitvektor des Agenten im ATAC
d	Richtungsattribut eines Agenten
e	Agentenentscheidung
F	Flag im ATAC
i	Anzahl der möglichen Inputkombinationen einer FSM
k	Anzahl der Agenten
m	Bewegungsbedingung für einen Agenten
n	Anzahl der möglichen Zustände einer FSM
o	Anzahl der möglichen Outputkombinationen einer FSM
P	Prioritätenliste zur Konfliktauflösung
pos	Positionsfunktion der Prioritätenliste
s	Kontrollzustand des Agenten
w	Swap-Bedingung eines Agenten (in der Zellregel)
x	Input für die Kontrollfunktion des Agenten
x_r	reduzierter Input für die Kontrollfunktion des Agenten
y	Output der Kontrollfunktion des Agenten
Δx, Δy	horizontale/vertikale Distanz im zweidimensionalen Zellfeld
τ	Zelltyp

Heuristische Optimierungsverfahren

C	Menge der Heruasforderer (engl.: Challengers) für die Wettbewerbsselektion
D	Menge der Verteidiger (engl.: Defenders) für die Wettbewerbsselektion
f	Fitness einer Lösung
I	Anzahl der Inseln im Insellmodell-GA
K	Menge aller Lösungskandidaten im Suchraum
M	Mutationstechnik
O	Menge der Nachkommen (engl.: Offspring) im GA
p	Mutations-, Rekombinations- oder Selektionswahrscheinlichkeiten
P	Population in einem evolutionären Algorithmus
q	Elternquote für die Selektion
Q	Q-Wert beim Q-Learning
r	Belohnungsfunktion beim Q-Learning
R	Rekombinationstechnik
S	Menge der für die Rekombination selektierten Lösungen
t	diskreter Zeitpunkt (Iteration) im GA oder Berechnungszeit der Evolvierung
W	Menge der Gewinner (engl.: Winners) der Wettbewerbsselektion
α	Lernrate beim Q-Learning
γ	Discountfaktor beim Q-Learning

Optimierungstechniken für die Agentenstruktur

ce	Clock enable für Time-Shuffling und randomisierte FSMs
p	Zufallswahrscheinlichkeit bei randomisierten FSMs
T	Time-Shuffling-Periode
λ	Time-Shuffling-Selektor
Π	Anzahl der möglichen TS-Perioden bzw. Zufallswahrscheinlichkeiten

Qualitätsbewertung der Kontrollfunktionen

A	Abdeckung der Zellen im CEP
ack	Acknowledge-Signal für Informationsgewinn im ATAC
C	Kosten für die Lösung einer Welt
c_v	Anzahl der vollständig informierten Agenten im ATAC
com	Indikator für stattgefundene Kommunikation im ATAC
E	Anzahl der erfolgreich gelösten Welten
G	mittlere Dauer für die Lösung einer Welt
$\check{g}$	Anzahl der Generationen, die für einen Erfolg benötigt werden
j	Index der zu lösenden Agentenwelten
J	Anzahl der zu lösenden Agentenwelten
lim	Grenze, ab der eine Strafe zur Fitness addiert wird
pen	Fitnessstrafe für das Nicht-Erreichen eines Mindestziels
suc	Indikator für eine erfolgreich gelöste Welt
t, $\bar{t}$	erreichte/nicht erreichte Zielpositionen im ARP
v, $\bar{v}$	besuchte/nicht besuchte Zellen im CEP
Z	Anzahl aller zu besuchenden Zellen im CEP

Literaturverzeichnis

[AAAB09] Al-Sawi, Yazed ; Al-Ajlan, Ajlan ; Al-Drewiesh, Khalid ; Bajahzer, Abdullah: An Efficient Search Agent Software for Multi-Agent Systems Using Finite State Machine Technology. In: *International Journal of Digital Content Technology and its Applications* 3 (2009), März, Nr. 1, S. 73–87

[Ada98] Adami, Christoph: *Introduction to Artificial Life*. TELOS, 1998

[AHY+10] Alexander, George ; Heckel, Frederick W. P. ; Youngblood, G. M. ; Hale, D. H. ; Ketkar, Nikhil S.: Rapid Development of Intelligent Agents in First/Third-Person Training Simulations via Behavior-Based Control. In: *Proceedings of the 19th Conference on Behavior Representation in Modeling and Simulation, Charleston, SC, 21-24 March*, 2010, S. 187–194

[And06] Anderson, Carl: Creation of Desirable Complexity: Strategies for Designing Selforganized Systems. In: Braha, Dan (Hrsg.) ; Minai, Ali A. (Hrsg.) ; Bar-Yam, Yaneer (Hrsg.): *Complex Engineered Systems* Bd. 14. Springer Berlin / Heidelberg, 2006, 101–121

[Ari07] Aristoteles ; Wolf, Ursula (Hrsg.): *Metaphysik 1041b 10 (VII. Buch (Z))*. 5. Auflage. Rowohlt, 2007. – übersetzt von Hermann Bönitz

[AS07] Amigoni, Francesco ; Schiaffonati, Viola: Multiagent-Based Simulation in Biology. In: Magnani, Lorenzo (Hrsg.) ; Li, Ping (Hrsg.): *Model-Based Reasoning in Science, Technology, and Medicine* Bd. 64. Springer, 2007, 179–191

[BA02] Bordogna, C. M. ; Albano, E. V.: A Cellular Automata Model for Social-learning Processes in a Classroom Context . In: *The European Physical Journal B - Condensed Matter and Complex Systems* 25 (2002), Nr. 3, S. 391–396

[Ban99] Bandman, Olga L.: Comparative Study of Cellular-Automata Diffusion Models. In: Malyshkin, Victor E. (Hrsg.): *Parallel Computing Technologies, 5th International Conference, PaCT-99, St. Petersburg, Russia, September 6-10, 1999, Proceedings* Bd. 1662, Springer, 1999 (Lecture Notes in Computer Science), 395–409

[Bat94] Bates, Joseph: The Role of Emotion in Believable Agents. In: *Communications of the ACM* 37 (1994), Juli, Nr. 7, S. 122–125

[BB97] Bonarini, Andrea ; Basso, Filippo: Learning to Compose Fuzzy Behaviors for Autonomous Agents. In: *Int. Journal of Approximate Reasoning* 17 (1997), Nr. 4, 409–432

[BB05] Bäck, Thomas ; Breukelaar, Ron: Using Genetic Algorithms to Evolve Behavior in Cellular Automata. In: Calude, Cristian S. (Hrsg.) ; Dinneen, Michael J. (Hrsg.) ; Paun, Gheorghe (Hrsg.) ; Pérez-Jiménez, Mario J. (Hrsg.) ; Rozenberg, Grzegorz

(Hrsg.): *Unconventional Computing, 4th International Conference, UC 2005, Sevilla, Spain, October 3-7, 2005, Proceedings* Bd. 3699, 2005 (Lecture Notes in Computer Science), S. 1–10

[BCHM06] Bernon, Carole ; Chevrier, Vincent ; Hilaire, Vincent ; Marrow, Paul: Applications of Self-Organising Multi-Agent Systems: An Initial Framework for Comparison. In: *Informatica (Slovenia)* 30 (2006), Nr. 1, 73–82

[BDT99] Bonabeau, Eric ; Dorigo, Marco C. ; Theraulaz, Guy: *Swarm Intelligence: From Natural to Artificial Systems*. Oxford : Oxford University Press, 1999

[Bez99] Bezzi, Michele: *Modeling Biology by Cellular Automata*, Dipartamento di Fisica, Università di Bologna, Diss., 1999

[BHW01] Busch, Costas ; Herlihy, Maurice ; Wattenhofer, Roger: Routing Without Flow Control. In: *SPAA*. New York : ACM Press, 2001, S. 11–20

[BL04] Bousquet, F. ; Le Page, C.: Multi-Agent Simulations and Ecosystem Management: A Review. In: *Ecological Modelling* 176 (2004), September, Nr. 3-4, S. 313–332

[BMM+08] Bandini, Stefania ; Manzoni, Sara ; Mauri, Giancarlo ; Redaelli, Stefano ; Vanneschi, Leonardo: GP Generation of Pedestrian Behavioral Rules in an Evacuation Model Based on SCA. In: Umeo, Hiroshi (Hrsg.) ; Morishita, Shin (Hrsg.) ; Nishinari, Katsuhiro (Hrsg.) ; Komatsuzaki, Toshihiko (Hrsg.) ; Bandini, Stefania (Hrsg.): *Cellular Automata, 8th International Conference on Cellular Aotomata for Reseach and Industry, ACRI 2008, Yokohama, Japan, September 23-26, 2008. Proceedings* Bd. 5191, Springer, 2008 (Lecture Notes in Computer Science), 409–416

[BMS02] Bandini, Stefania ; Manzoni, Sara ; Simone, Carla: Enhancing Cellular Spaces by Multilayered Multi Agent Situated Systems. In: Bandini, Stefania (Hrsg.) ; Chopard, Bastien (Hrsg.) ; Tomassini, Marco (Hrsg.): *ACRI* Bd. 2493, Springer, 2002 (Lecture Notes in Computer Science), 156–167

[BMV09] Bandini, Stefania ; Manzoni, Sara ; Vizzari, Giuseppe: Agent Based Modeling and Simulation. In: Meyers, Robert A. (Hrsg.): *Encyclopedia of Complexity and Systems Science*. Springer, 2009, S. 184–197

[Bra93] Braitenberg, Valentino: *Vehikel: Experimente mit kybernetischen Wesen*. Rohwolt, 1993

[BS04] Bourg, David M. ; Seamann, Glenn: *AI for Game Developers*. O'Reilly, 2004

[BS08] *Kapitel* 6. In: Branke, Jürgen ; Schmeck, Hartmut: *Evolutionary Design of Emergent Behavior*. Springer, 2008, S. 123–140

[BSW85] Beyer, William A. ; Sellers, Peter H. ; Waterman, Michael S.: Stanislaw M. Ulam's Contributions to Theoretical Theory. In: *Letters in Mathematical Physics* 10 (1985), S. 231–242

[BT95] Blickle, Tobias ; Thiele, Lothar: A Comparison of Selection Schemes used in Genetic Algorithms / Swiss Federal Institute of Technology (ETH) Zürich. 1995 (11, Version 2). – TIK-Report

[Buc05] BUCKLAND, Mat: *Programming Game AI by Example*. Wordware Publishing, 2005

[BZN04] BARBAT, Boldur E. ; ZAMFIRESCU, Constantin B. ; NEGULESCU, Sorin C.: The Best from Ants and Humans: Synergy in Agent-Based Systems. In: *Studies in Informatics and Control* 13 (2004), März, Nr. 1, S. 47–59

[CGHH89] COHEN, Paul R. ; GREENBURG, Michael L. ; HART, David M. ; HOWE, Adele E.: Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. In: *AI Magazine* 10 (1989), Nr. 3, S. 32–48

[CJ90] COLLINS, Robert J. ; JEFFERSON, David R.: An Artificial Neural Network Representation for Artificial Organisms. In: SCHWEFEL, H.-P. (Hrsg.) ; MÄNNER, R. (Hrsg.): *Parallel Problem Solving from Nature* Bd. 496. Berlin : Springer, 1990 (Lecture Notes in Computer Science), S. 259–263

[CJ91] COLLINS, Robert J. ; JEFFERSON, David R.: AntFarm: Towards Simulated Evolution. In: LANGTON, Christopher G. (Hrsg.) ; TAYLOR, Charles (Hrsg.) ; FARMER, J. D. (Hrsg.) ; RASMUSSEN, Steen (Hrsg.): *Artificial Life II* Bd. 5. Redwood City, CA, USA : Addison-Wesley, Februar 1991 (Santa Fe Institute Studies in the Sciences of Complexity), S. 579–602

[Cod68] CODD, E. F.: *Cellular Automata*. New York : Academic Press, 1968

[Cri02] CRICHTON, Michael: *Prey*. Harper, 2002

[CW06] CORREIA, Luís ; WEHRLE, Thomas: Beyond Cellular Automata, Towards More Realistic Traffic Simulators. In: YACOUBI, Samira E. (Hrsg.) ; CHOPARD, Bastien (Hrsg.) ; BANDINI, Stefania (Hrsg.): *ACRI* Bd. 4173, Springer, 2006 (Lecture Notes in Computer Science), 690–693

[DD97] DI CARO, Gianni ; DORIGO, Marco: AntNet: A Mobile Agents Approach to Adaptive Routing / IRIDIA, Université Libre de Bruxelles, Belgium. 1997 (97-12). – Forschungsbericht

[DD99] DORIGO, Marco ; DI CARO, Gianni: Ant Colony Optimization: A New Meta-Heuristic. In: ANGELINE, Peter J. (Hrsg.) ; MICHALEWICZ, Zbyszek (Hrsg.) ; SCHOENAUER, Marc (Hrsg.) ; YAO, Xin (Hrsg.) ; ZALZALA, Ali (Hrsg.): *Proceedings of the Congress on Evolutionary Computation* Bd. 2, IEEE Press, 1999, S. 1470–1477

[DFH+04] DI MARZO SERUGENDO, Giovanna ; FOUKIA, Noria ; HASSAS, Salima ; KARAGEORGOS, Anthony ; KOUADRI MOSTÉFAOUI, Soraya ; RANA, Omer F. ; ULIERU, Mihaela ; VALCKENAERS, Paul ; VAN AART, Chris: Self-Organisation: Paradigms and Applications. In: DI MARZO SERUGENDO, G. (Hrsg.) ; KARAGEORGOS, A. (Hrsg.) ; RANA, O. F. (Hrsg.) ; ZAMBONELLI, F. (Hrsg.): *Proceedings of the Engineering Self-Organising Applications Workshop (ESOA'03)*, Springer, 2004 (LNCS), S. 1–19

[DGK05] DI MARZO SERUGENDO, Giovanna ; GLEIZES, Marie-Pierre ; KARAGEORGOS, Anthony: Self-organization in Multi-Agent Systems. In: *Knowledge Eng. Review* 20 (2005), Nr. 2, S. 165–189

[DGK06] Di Marzo Serugendo, Giovanna ; Gleizes, Marie-Pierre ; Karageorgos, Anthony: Self-organisation and Emergence in MAS: An Overview. In: *Informatica* 30 (2006), S. 45–54

[DH04] De Wolf, Tom ; Holvoet, Tom: Emergence and Self-Organisation: A Statement of Similarities and Differences. In: *Second International Workshop on Engineering Self-Organising Applications (ESOA)*, Springer, 2004 (Lecture Notes in Artificial Intelligence), S. 96–110

[DH05a] De Wolf, Tom ; Holvoet, Tom: Emergence Versus Self-Organisation: Different Concepts But Promising When Combined. In: *Engineering Self-Organising Systems* Bd. 3464, Springer-Verlag, 2005 (Lecture Notes in Computer Science), S. 1–15

[DH05b] De Wolf, Tom ; Holvoet, Tom: Towards a Full Life-cycle Methodology for Engineering Decentralised Multi-Agent Systems. In: Gonzalez-Perez, C. (Hrsg.): *Proceedings of the Fourth International Workshop on Agent-Oriented Methodologies*, 2005, 1–12

[DJT01] Dijkstra, J. ; Jessurun, J. ; Timmermans, Harry J. P.: A Multi-Agent Cellular Automata Model of Pedestrian Movement. In: Schreckenberg, M. (Hrsg.) ; Sharma, S. D. (Hrsg.): *Pedestrian and Evacuation Dynamics*, Springer, 2001, S. 173–181

[DK84] Domany, Eytan ; Kinzel, Wolfgang: Equivalence of Cellular Automata to Ising Models and Directed Percolation. In: *Physical Review Letters* 53 (1984), Nr. 4, S. 311–314

[DL06] Di Stefano, Bruno N. ; Lawniczak, Anna T.: Autonomous Roving Object's Coverage of its Universe. In: *CCECE*, IEEE, 2006, 1591–1594

[DLC89] Durfee, Edmund H. ; Lesser, Victor R. ; Corkill, Daniel D.: Trends in Cooperative Distributed Problem Solving. In: *IEEE Transactions on Knowledge and Data Engineering* 1 (1989), März, Nr. 1, S. 63–83

[DV07] Dhillon, S.S. ; Van Mieghem, P.: Performance Analysis of the AntNet Algorithm. In: *Computer Networks* 51 (2007), Nr. 8, 2104 - 2125

[EA96] Epstein, Joshua M. ; Axtell, Robert L.: *Growing Artificial Societies: Social Science from the Bottom Up*. Washington, D.C. : Brookings Institution Press, 1996

[Edm04] Edmonds, Bruce: Using the Experimental Method to Produce Reliable Self-Organised Systems. In: *Engineering Self Organising Sytems: Methodologies and Applications. Volume 3464 of Lecture Notes in Artificial Intelligence*, Springer, 2004, S. 84–99

[EH08a] Ediger, Patrick ; Hoffmann, Rolf: Improving the Behavior of Creatures by Time-Shuffling. In: Umeo, Hiroshi (Hrsg.) ; Morishita, Shin (Hrsg.) ; Nishinari, Katsuhiro (Hrsg.) ; Komatsuzaki, Toshihiko (Hrsg.) ; Bandini, Stefania (Hrsg.): *Cellular Automata, 8th International Conference on Cellular Automata for Reseach and Industry, ACRI 2008, Yokohama, Japan, September 23-26, 2008. Proceedings* Bd. 5191, Springer, 2008 (Lecture Notes in Computer Science), 345–353

[EH08b] EDIGER, Patrick ; HOFFMANN, Rolf: Optimizing the Creature's Rule for All-to-All Communication. In: *EPSRC Workshop Automata-2008. Theory and Applications of Cellular Automata, Bristol, UK, June 12-14, 2008*, 2008, S. 398–410

[EH09a] EDIGER, Patrick ; HOFFMANN, Rolf: Alignment of Particles by Agents with Evolved Behaviour. In: *1. Workshop Innovative Rechnertechnologien, 9. - 10. Juli 2009, Hamburg*, 2009

[EH09b] EDIGER, Patrick ; HOFFMANN, Rolf: CA Models for Target Searching Agents. In: *EPSRC Workshop Automata-2009. Theory and Applications of Cellular Automata, São José dos Campos, Brazil, October 10-12, 2009*, 2009, S. 41–54

[EH09c] EDIGER, Patrick ; HOFFMANN, Rolf: Combining Genetic Algorithm with Time-Shuffling in Order to Evolve Agent Systems More Efficiently. In: IPDPS PROCEEDINGS (Hrsg.): *Proc. 12th International Workshop on Nature Inspired Distributed Computing (NIDISC), 23rd IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, IEEE, 2009

[EH09d] EDIGER, Patrick ; HOFFMANN, Rolf: Solving All-to-All Communication with CA Agents More Effectively with Flags. In: MALYSHKIN, Victor (Hrsg.): *Parallel Computing Technologies, 10th International Conference, PaCT 2009, Novosibirsk, Russia, August 31-September 4, 2009. Proceedings* Bd. 5698, 2009 (LNCS), S. 182–193

[EH10a] EDIGER, Patrick ; HOFFMANN, Rolf: All-to-All Communication with CA Agents by Active Coloring and Acknowledging. In: BANDINI, Stefania (Hrsg.) ; MANZONI, S. (Hrsg.) ; UMEO, Hiroshi (Hrsg.) ; VIZZARI, G. (Hrsg.): *Cellular Automata, 9th International Conference on Cellular Automata for Reseach and Industry, ACRI 2010, Ascoli Piceno, Italy, September 21-24, 2010. Proceedings* Bd. 6350. Heidelberg : Springer, 2010 (Lecture Notes in Computer Science), S. 24–34

[EH10b] EDIGER, Patrick ; HOFFMANN, Rolf: Evolving Hybrid Time-Shuffled Behavior of Agents. In: *Proc. 13th International Workshop on Nature Inspired Distributed Computing (NIDISC), 24th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2010

[EH10c] EDIGER, Patrick ; HOFFMANN, Rolf: Evolving Probabilistic CA Agents Solving the Routing Task. In: *16th International Workshop on Cellular Automata and Discrete Complex Systems, AUTOMATA, Nancy, France, June 14-16, 2010*, 2010

[EH10d] EDIGER, Patrick ; HOFFMANN, Rolf: Routing Based on Evolved Agents. In: *23rd PARS Workshop on Parallel Systems and Algorithms, Hannover, Germany*, 2010, S. 45–53

[EHD10] EDIGER, Patrick ; HOFFMANN, Rolf ; DÉSÉRABLE, Dominique: Routing in the Triangular Grid with Evolved Agents. In: SMARI, Waleed W. (Hrsg.): *Proceedings of the 2010 International Conference on High Performance Computing and Simulation (HPCS), June 28 - July 2, 2010, Caen, France*, 2010, S. 582–590

[EHG11] EDIGER, Patrick ; HOFFMANN, Rolf ; GRÜNER, Sylvia: Effectively Evolving Finite State Machines Compared to Enumeration. In: QUESADA-ARENCIBIA, Alexis (Hrsg.) ; RODRIGUEZ, José C. (Hrsg.) ; MORENO-DÍAZ JR., Roberto (Hrsg.) ; MORENO-DÍAZ, Roberto

(Hrsg.): *EUROCAST 2011, 13th International Conference on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 2011, Extended Abstracts*, 2011, S. 267–268

[EHH08a] EDIGER, Patrick ; HOFFMANN, Rolf ; HALBACH, Mathias: How Efficient are Creatures with Time-Shuffled Behaviors? In: NAGEL, Wolfgang E. (Hrsg.) ; HOFFMANN, Rolf (Hrsg.) ; KOCH, Andreas (Hrsg.): *9th Workshop on Parallel Systems and Algorithms (PASA) held at the 21st Conference on the Architecture of Computing Systems (ARCS), February 26th, 2008, in Dresden, Germany* Bd. 124, GI, 2008 (Lecture Notes in Informatics), S. 93–103

[EHH08b] EDIGER, Patrick ; HOFFMANN, Rolf ; HALBACH, Mathias: Is a Non-Uniform System of Creatures more Efficient than a Uniform One? In: IPDPS PROCEEDINGS (Hrsg.): *Proc. 11th International Workshop on Nature Inspired Distributed Computing (NIDISC), 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, IEEE, 2008

[EHH09] EDIGER, Patrick ; HOFFMANN, Rolf ; HALBACH, Mathias: Evolving 6-state Automata for Optimal Behaviors of Creatures Compared to Exhaustive Search. In: MORENO-DÍAZ, Roberto (Hrsg.) ; PICHLER, Franz (Hrsg.) ; QUESADA-ARENCIBIA, Alexis (Hrsg.): *Computer Aided Systems Theory - EUROCAST 2009, 12th International Conference, Las Palmas de Gran Canaria, Spain, February 15-20, 2009, Revised Selected Papers* Bd. 5717, Springer, 2009 (Lecture Notes in Computer Science), S. 689–696

[Elm90] ELMAN, Jeffrey L.: Finding Structure in Time. In: *Cognitive Science* 14 (1990), April, Nr. 2, S. 179–211

[ES03] EIBEN, A. E. ; SMITH, J. E.: *Introduction to Evolutionary Computing*. Springer-Verlag, 2003 (Natural Computing Series)

[FG96] FRANKLIN, Stan ; GRAESSER, Arthur C.: Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. In: MÜLLER, Jörg P. (Hrsg.) ; WOOLDRIDGE, Michael (Hrsg.) ; JENNINGS, Nicholas R. (Hrsg.): *Proceedings of the ECAI'96 Workshop on Agent Theories, Architectures, and Languages: Intelligent Agents III* Bd. 1193, Springer, 1996 (Lecture Notes in Computer Science), S. 21–35

[FKSL07] FEY, Dietmar ; KOMANN, Marcus ; SCHURZ, Frank ; LOOS, Andreas: An Organic Computing Architecture for Visual Microprocessors Based on Marching Pixels. In: *ISCAS*, IEEE, 2007, 2686–2689

[FS05] FEY, Dietmar ; SCHMIDT, Daniel: Marching Pixels: A New Organic Computing Paradigm for Smart Sensor Processor Arrays. In: BAGHERZADEH, Nader (Hrsg.) ; VALERO, Mateo (Hrsg.) ; RAMÍREZ, Alex (Hrsg.): *Conf. Computing Frontiers*, ACM, 2005, 1–9

[GADP89] GOSS, S. ; ARON, S. ; DENEUBOURG, J. L. ; PASTEELS, J. M.: Self-organized Shortcuts in the Argentine Ant. In: *Naturwissenschaften* 76 (1989), Nr. 12, S. 579–581

[Gar70] GARDNER, Martin: The Fantastic Combinations of John Conway's New Solitaire Game „Life". In: *Scientific American* 223 (1970), Oktober, Nr. 10, S. 120–123

[Gar83] GARDNER, Martin: *Wheels, Life, and Other Mathematical Amusements*. New York : W. H. Freeman, 1983

[GGT07] GARNIER, Simon ; GAUTRAIS, Jacques ; THERAULAZ, Guy: The Biological Principles of Swarm Intelligence. In: *Swarm Intelligence* 1 (2007), Nr. 1, S. 3–31

[GK94] GENESERETH, Michael R. ; KETCHPEL, Steven P.: Software Agents. In: *Communications of the ACM, Special Issue on Intelligent Agents* 37 (1994), Juli, Nr. 7, S. 48–53

[Glo90] GLOVER, Fred: Tabu Search: A Tutorial. In: *Interfaces* 20 (1990), Nr. 4, S. 74–94

[GMW94] GORDON, V. S. ; MATHIAS, Keith ; WHITLEY, Darrell: Cellular Genetic Algorithms as Function Optimizers: Locality Effects. In: *SAC '94: Proceedings of the 1994 ACM symposium on Applied computing*. New York, NY, USA : ACM, 1994, S. 237–241

[GN89] GENESERETH, Michael R. ; NILSSON, Nils J.: *Logische Grundlagen der Künstlichen Intelligenz*. Vieweg, 1989

[Gol89] GOLDBERG, David E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA : Addison-Wesley, 1989

[Gor07] GORDON, Deborah M.: Control without Hierarchy. In: *Nature* 446 (2007), März, S. 143

[Gru09] GRUNER, Stefan: Mobile Agent Systems and Cellular Automata. In: *Autonomous Agents and Multi-Agent Systems* (2009)

[GSD+03] GANGULY, Niloy ; SIKDAR, Biplab K. ; DEUTSCH, Andreas ; CANRIGHT, Geoffrey ; CHAUDHURI, P. P.: A Survey on Cellular Automata / Centre for High Performance Computing, Dresden University of Technology. 2003 (30). – Forschungsbericht

[Had77] HADLOCK, F. O.: A Shortest Path Algorithm for Grid Graphs. In: *Networks* 7 (1977), S. 323–334

[Hal08] HALBACH, Mathias: *Algorithmen und Hardwarearchitekturen zur optimierten Aufzählung von Automaten und deren Einsatz bei der Simulation künstlicher Kreaturen*, Technische Universität Darmstadt, Diss., 2008

[HBC+00] HOLLAND, John H. ; BOOKER, Lashon B. ; COLOMBETTI, Marco ; DORIGO, Marco ; GOLDBERG, David E. ; FORREST, Stephanie ; RIOLO, Rick L. ; SMITH, Robert E. ; LANZI, Pier L. ; STOLZMANN, Wolfgang ; WILSON, Stewart W.: What Is a Learning Classifier System? In: LANZI, Pier L. (Hrsg.) ; STOLZMANN, Wolfgang (Hrsg.) ; WILSON, Stewart W. (Hrsg.): *Learning Classifier Systems: From Foundations to Applications* Bd. 1813, Springer, 2000 (LNAI), S. 3–32

[HCPA07] HERRERO, Álvaro ; CORCHADO, Emilio ; PELLICER, María A. ; ABRAHAM, Ajith: Hybrid Multi Agent-Neural Network Intrusion Detection with Mobile Visualization. In: CORCHADO, Emilio (Hrsg.) ; CORCHADO, Juan M. (Hrsg.) ; ABRAHAM, Ajith (Hrsg.): *Innovations in Hybrid Intelligent Systems* Bd. 44. Springer, 2007, 320–328

[HE08] Hoffmann, Rolf ; Ediger, Patrick: Evolving Multi-creature Systems for All-to-All Communication. In: Umeo, Hiroshi (Hrsg.) ; Morishita, Shin (Hrsg.) ; Nishinari, Katsuhiro (Hrsg.) ; Komatsuzaki, Toshihiko (Hrsg.) ; Bandini, Stefania (Hrsg.): *Cellular Automata, 8th International Conference on Cellular Automata for Reseach and Industry, ACRI 2008, Yokohama, Japan, September 23-26, 2008. Proceedings* Bd. 5191, Springer, 2008 (Lecture Notes in Computer Science), 550–554

[Hee07] Heenes, Wolfgang: *Entwurf und Realisierung von massivparallelen Architekturen für Globale Zellulare Automaten*, Technische Universität Darmstadt, Diss., 2007

[Her07] Herrler, Rainer: *Agentenbasierte Simulation zur Ablaufoptimierung in Krankenhäusern und anderen verteilten, dynamischen Umgebungen*, Universitätsbibliothek Würzburg; Fakultät für Mathematik und Informatik. Institut für Informatik, Diss., 2007

[HH05] Halbach, Mathias ; Hoffmann, Rolf: Optimal Behavior of a Moving Creature in the Cellular Automata Model. In: Malyshkin, Victor E. (Hrsg.): *Parallel Computing Technologies, 8th International Conference, PaCT 2005, Krasnoyarsk, Russia, September 5-9, 2005, Proceedings* Bd. 3606, Springer, 2005 (Lecture Notes in Computer Science), 129–140

[HH06a] Halbach, Mathias ; Hoffmann, Rolf: Minimising the Hardware Resources for a Cellular Automaton with Moving Creatures. In: Karl, Wolfgang (Hrsg.) ; Becker, Jürgen (Hrsg.) ; Grosspietsch, Karl-Erwin (Hrsg.) ; Hochberger, Christian (Hrsg.) ; Maehle, Erik (Hrsg.): *ARCS 2006 - 19th International Conference on Architecture of Computing Systems, Workshops Proceedings, March 16, 2006, Frankfurt am Main, Germany* Bd. 81, GI, 2006 (LNI), S. 323–332

[HH06b] Hoffmann, Rolf ; Halbach, Mathias: Are Several Creatures More Efficient Than a Single One? In: El Yacoubi, Samira (Hrsg.) ; Chopard, Bastien (Hrsg.) ; Bandini, Stefania (Hrsg.): *Cellular Automata, 7th International Conference on Cellular Automata, for Research and Industry, ACRI 2006, Perpignan, France, September 20-23, 2006, Proceedings* Bd. 4173, Springer, 2006 (Lecture Notes in Computer Science), 707–711

[HH07a] Halbach, Mathias ; Hoffmann, Rolf: Parallel Hardware Architecture to Simulate Movable Creatures in the CA Model. In: Malyshkin, Victor E. (Hrsg.): *Parallel Computing Technologies, 9th International Conference, PaCT 2007, Pereslavl-Zalessky, Russia, September 3-7, 2007, Proceedings* Bd. 4671, Springer, 2007 (LNCS), S. 418–431

[HH07b] Halbach, Mathias ; Hoffmann, Rolf: Solving the Exploration's Problem with Several Creatures More Efficiently. In: Moreno-Díaz, Roberto (Hrsg.) ; Pichler, Franz (Hrsg.) ; Quesada-Arencibia, Alexis (Hrsg.): *Computer Aided Systems Theory - EUROCAST 2007, 11th International Conference on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 12-16, 2007, Revised Selected Papers* Bd. 4739, Springer, 2007 (Lecture Notes in Computer Science), 596–603

[HHB06] Halbach, Mathias ; Hoffmann, Rolf ; Both, Lars: Optimal 6-State Algorithms for the Behavior of Several Moving Creatures. In: El Yacoubi, Samira (Hrsg.) ; Chopard,

Bastien (Hrsg.) ; BANDINI, Stefania (Hrsg.): *Cellular Automata, 7th International Conference on Cellular Automata, for Research and Industry, ACRI 2006, Perpignan, France, September 20-23, 2006, Proceedings* Bd. 4173, Springer, 2006 (Lecture Notes in Computer Science), 571–581

[HHHT04] HALBACH, Mathias ; HEENES, Wolfgang ; HOFFMANN, Rolf ; TISJE, Jan: Optimizing the Behavior of a Moving Creature in Software and in Hardware. In: SLOOT, Peter M. (Hrsg.) ; CHOPARD, Bastien (Hrsg.) ; HOEKSTRA, Alfons G. (Hrsg.): *Cellular Automata, 6th International Conference on Cellular Automata for Research and Industry, ACRI 2004, Amsterdam, The Netherlands, October 25-28, 2004, Proceedings* Bd. 3305, Springer, 2004 (Lecture Notes in Computer Science), S. 841–850

[HHW99] HOCHBERGER, Christian ; HOFFMANN, Rolf ; WALDSCHMIDT, Stefan: CDL++ for the Description of Moving Objects in Cellular Automata. In: MALYSHKIN, Victor E. (Hrsg.): *Parallel Computing Technologies, 5th International Conference, PaCT-99, St. Petersburg, Russia, September 6-10, 1999, Proceedings* Bd. 1662, Springer, 1999 (Lecture Notes in Computer Science), 428–435

[HL73] HERMAN, G. T. ; LIU, W. H.: The Daughter of CELIA, the French Flag and the Firing Squad. In: *WSC '73: Proceedings of the 6th conference on Winter simulation*. New York, NY, USA : ACM, 1973, 870

[Hoc98] HOCHBERGER, Christian: *CDL - Eine Sprache für die Zellularverarbeitung auf verschiedenen Zielplattformen*, Technische Universität Darmstadt, Diss., 1998

[Hof93] HOFFMANN, Rolf: *Rechnerentwurf: Rechenwerke, Mikroporgrammierung, RISC*. 3. Auflage. Oldenbourg, 1993

[Hol00] HOLLAND, John H.: *Emergence: From Chaos to Order*. Oxford University Press, 2000

[HR95] HAYES-ROTH, Barbara: An Architecture for Adaptive Intelligent Systems. In: *Artificial Intelligence* 72 (1995), Januar, Nr. 1-2, S. 329–365

[HRT+05] HINCHEY, Michael G. ; RASH, James L. ; TRUSZKOWSKI, Walter F. ; ROUFF, Christopher A. ; STERRITT, Roy: Autonomous and Autonomic Swarms. In: *In Proceedings of Autonomic & Autonomous Space Exploration Systems (A&A-SES-1) at 2005 International Conference on Software Engineering Research and Practice (SERP'05), Las Vegas, NV*, CREA Press, 2005, S. 27–30

[HS99] HUHNS, Michael N. ; STEPHENS, Larry M.: Multiagent Systems and Societies of Agents. In: WEISS, Gerhard (Hrsg.): *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA : The MIT Press, 1999, Kapitel 2, S. 79–120

[HS03] HOFFMANN, Rolf ; SAMBHARAJU, Smitha: Optimal Algorithm for Checking the Environment by a Moving Creature / Technische Universität Darmstadt. 2003 (RA-1-2003). – Forschungsbericht

[HUVW01] HOFFMANN, Rolf ; ULMANN, Bernd ; VÖLKMANN, Klaus-Peter ; WALDSCHMIDT, Stefan: The Machine CEPRA-S Configured for Stream Processing. In: *9th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey (CA), 11-13 Feb. 2001*, 2001

[HVW00] Hoffmann, Rolf ; Völkmann, Klaus-Peter ; Waldschmidt, Stefan: Global Cellular Automata GCA: An Universal Extension of the CA Model. In: *ACRI 2000 "work in progress" session, Karlsruhe, Germany, Oct. 4th - 6th*, 2000

[IN97] Ikegami, Takashi ; Nishimura, Shin I.: Emergence of Collective Strategies in a Prey-Predator Game Model. In: *Artificial Life* 3 (1997), Nr. 4, S. 243–260

[JCC+90] Jefferson, David ; Collins, Robert ; Cooper, Claus ; Dyer, Michael ; Flowers, Margot ; Korf, Richard ; Taylor, Charles ; Wang, Alan: Evolution as a Theme in Artificial Life: The Genesys/Tracker System / University of California, Los Angeles. 1990 (UCLA-AI-90-09). – Forschungsbericht

[JM08] Jaafar, Jafreezal ; McKenzie, Eric: A Fuzzy Action Selection Method for Virtual Agent Navigation in Unknown Virtual Environments. In: *Journal of Uncertain Systems* 2 (2008), Nr. 2, S. 144–154

[JSW98] Jennings, N. ; Sycara, K. ; Wooldridge, M.: A Roadmap of Agent Research and Development. In: *Autonomous Agents and Multi-Agent Systems* 1 (1998), Nr. 1, 7–38

[KC03] Kephart, Jeffrey O. ; Chess, David M.: The Vision of Autonomic Computing. In: *IEEE Computer* 36 (2003), Nr. 1, 41–50

[KCS00] Kier, L. B. ; Cheng, C. K. ; Seybold, P. G.: Cellular Automata Models of Chemical Systems. In: *SAR and QSAR in Environmental Research* 11 (2000), Nr. 2, S. 79–102

[KEFH09] Komann, Marcus ; Ediger, Patrick ; Fey, Dietmar ; Hoffmann, Rolf: On the Effectivity of Genetic Programming Compared to the Time-Consuming Full Search of Optimal 6-State Automata. In: Vanneschi, Leonardo (Hrsg.) ; Gustafson, Steven (Hrsg.) ; Ebner, Marc (Hrsg.): *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*. Tuebingen : Springer, April 15-17 2009 (LNCS)

[KES01] Kennedy, James ; Eberhart, Russell C. ; Shi, Yuhi: *Swarm Intelligence*. San Francisco : Morgan Kaufman, 2001 (Evolutionary Computation Series)

[KF07] Komann, Marcus ; Fey, Dietmar: Realising Emergent Image Preprocessing Tasks in Cellular-Automaton-Alike Massively Parallel Hardware. In: *Int. J. Parallel Emerg. Distrib. Syst.* 22 (2007), Nr. 2, S. 79–89

[KF09] Komann, Marcus ; Fey, Dietmar: Evaluating the Evolvability of Emergent Agents with Different Numbers of States. In: Rothlauf, Franz (Hrsg.): *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009*, ACM, 2009, S. 1777–1778

[KF10] Komann, Marcus ; Fey, Dietmar: Revising the Trade-off between the Number of Agents and Agent Intelligence. In: Di Chio, Cecilia (Hrsg.) ; Cagnoni, Stefano (Hrsg.) ; Cotta, Carlos (Hrsg.) ; Ebner, Marc (Hrsg.) ; Ekárt, Anikó (Hrsg.) ; Esparcia-Alcazár, Anna I. (Hrsg.) ; Chi-Keong, Goh (Hrsg.) ; Merelo, Juan J. (Hrsg.) ; Neri, Ferrante (Hrsg.) ; Preuss, Mike (Hrsg.) ; Togelius, Julian (Hrsg.) ; Yannakakis, Georgios N. (Hrsg.): *Applications of Evolutionary Computation, EvoApplications 2010, Proceedings, Part I* Bd. 6024, 2010 (Lecture Notes in Computer Science), S. 31–40

[KFH04] KLÜGL, Franziska ; FEHLER, Manuel ; HERRLER, Rainer: About the Role of the Environment in Multi-Agent Simulations. In: WEYNS, Danny (Hrsg.) ; PARUNAK, H. Van D. (Hrsg.) ; MICHEL, Fabien (Hrsg.): *E4MAS* Bd. 3374, Springer, 2004 (Lecture Notes in Computer Science), 127–149

[KGV83] KIRKPATRICK, S. ; GELATT, JR., C. D. ; VECCHI, M. P.: Optimization by Simulated Annealing. In: *Science* 220 (1983), Nr. 4598, S. 671–680

[KLS07] KOENIG, Sven ; LIKHACHEV, Maxim ; SUN, Xiaoxun: Speeding up Moving-Target Search. In: DURFEE, Edmund H. (Hrsg.) ; YOKOO, Makoto (Hrsg.) ; HUHNS, Michael N. (Hrsg.) ; SHEHORY, Onn (Hrsg.): *AAMAS*, IFAAMAS, 2007, 1144–1151

[Klü01] KLÜGL, Franziska: *Multiagentensimulation - Konzepte, Werkzeuge, Anwendung*. Addison-Wesley, 2001

[Kom10] KOMANN, Marcus: *Manual Engineering and Evolution of Emergent Algorithms for Agents on Two-dimensional Grids*, Universität Erlangen-Nürnberg, Diss., 2010

[Koz92] KOZA, John R.: *Genetic Programming: On the Programming of Computers by Natural Selection*. Cambridge, MA : MIT Press, 1992

[Koz94] KOZA, John R.: *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, Massachusetts : The MIT Press, 1994

[Lan86] LANGTON, C G.: Studying Artificial Life With Cellular Automata. In: *Phys. D* 2 (1986), Nr. 1-3, S. 120–149

[Lan08] LANZI, Pier L.: Learning Classifier Systems: Then and Now. In: *Evolutionary Intelligence* 1 (2008), Nr. 1, 63–82

[LB95] LAND, Mark ; BELEW, Richard K.: No Perfect Two-State Cellular Automata for Density Classification Exists. In: *Phys. Rev. Lett.* 74 (1995), Jun, Nr. 25, S. 5148–5150

[LD09] LAWNICZAK, Anna T. ; DI STEFANO, Bruno N.: Development of Road Traffic CA Model of 4-Way Intersection to Study Travel Time. In: ZHOU, Jie (Hrsg.): *Complex Sciences, First International Conference, Complex 2009, Shanghai, China, February 23-25, 2009. Revised Papers, Part 2* Bd. 5, Springer, 2009 (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), 2040–2049

[Lee61] LEE, C. Y.: An Algorithm for Path Connection and Its Applications. In: *IRE Transactions on Electronic Computers* 10 (1961), Nr. 3, S. 346–365

[Lem67] LEM, Stanislaw: *Der Unbesiegbare*. Frankfurt am Main : Suhrkamp, 1967

[Lev93] LEVY, Steven: *KL - Künstliches Leben aus dem Computer*. Droemer Knaur, 1993

[LK05] LAU, Manfred ; KUFFNER, James J.: Behavior Planning for Character Animation. In: TERZOPOULOS, Demetri (Hrsg.) ; ZORDAN, Victor (Hrsg.): *ACM SIGGRAPH /Eurographics Symposium on Computer Animation*. Los Angeles, California : Eurographics Association, 2005, S. 271–280

[LK06] Lee, Mal ; Kang, Eung-Kwan: Learning Enabled Cooperative Agent Behavior in an Evolutionary and Competitive Environment. In: *Neural Computing and Applications* 15 (2006), Nr. 2, 124–135

[LMA05] Lin, J. ; Morse, A. S. ; Anderson, B. D. O.: The Multi-Agent Rendezvous Problem. An Extended Summary. In: *Cooperative Control* Bd. 309, Springer, 2005 (Lecture Notes in Computer Science), S. 257–289

[LP09] Loh, Peter K. K. ; Prakash, Edmond C.: Performance Simulations of Moving Target Search Algorithms. In: *Int. J. Comput. Games Technol.* 2009 (2009), S. 1–6

[MAE06] Mansour, Nashat ; Awad, Mohamad ; El-Fakih, Khaled: Incremental Genetic Algorithm. In: *The International Arab Journal of Information Technology* 3 (2006), Nr. 1, S. 42–47

[Mar93] Margolus, Norman: CAM-8: A Computer Architecture Based on Cellular Automata / MIT Laboratory For Computer Science. 1993 (01239). – Forschungsbericht

[Mar99] Marinescu, Dan C.: An Agent-Based Design for Problem Solving Environments. In: Bassetti, F. (Hrsg.) ; Davis, K. (Hrsg.) ; Mohr, B. (Hrsg.): *Proceedings of the Workshop on Parallel/High-Performance Object-Oriented Scientific Computing (POOSC'99)*, 1999, S. 24–34

[Mar04] Maron, Mikel: *How Adaptable are Swarms?* http://brainoff.com/easy/forage.pdf, 2004. – Project for Adaptive Systems, Evolutionary and Adaptive Systems, University of Sussex

[Mea55] Mealy, George H.: A Method for Synthesizing Sequential Circuits. In: *Bell System Technical Journal* 34 (1955), Nr. 5, S. 1045–1079

[Mer10] Merriam-Webster Online Dictionary: *agent*. http://www.merriam-webster.com/dictionary/agent, July 2010

[Mil99] Milojicic, Dejan S.: Trend Wars: Mobile Agent Applications. In: *IEEE Concurrency* 7 (1999), Juli/September, Nr. 3, 80–90

[Mit96] Mitchell, Melanie: *An Introduction to Genetic Algorithms*. Cambridge, MA : The MIT Press, 1996

[Mit97] Mitchell, Tom M.: *Machine Learning*. New York : McGraw Hill, 1997

[MM06] Müller-Schloer, C. ; Mnif, M.: Quantitative Emergence. In: *Proceedings of the 2006 IEEE Mountain Workshop on Adaptive and Learning Systems (IEEE SMCals 2006)*, 2006, S. 78–84

[MN05] Macal, Charles M. ; North, Michael J.: Tutorial on Agent-based Modeling and Simulation. In: *Winter Simulation Conference*, ACM, 2005, S. 2–15

[MN06] Macal, Charles M. ; North, Michael J.: Tutorial on Agent-based Modeling and Simulation Part 2: How to Model with Agents. In: Perrone, L. F. (Hrsg.) ; Lawson, Barry (Hrsg.) ; Liu, Jason (Hrsg.) ; Wieland, Frederick P. (Hrsg.): *Proceedings of the Winter Simulation Conference WSC 2006, Monterey, California, USA, December 3-6, 2006*, WSC, 2006, S. 73–83

[MOMC04] MELLO, Aline V. ; OST, Luciano C. ; MORAES, Fernando G. ; CALAZANS, Ney Laert V.: Evaluation of Routing Algorithms on Mesh Based NoCs / Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul. 2004 (040). – Forschungsbericht

[Moo56] MOORE, Edward F.: Gedanken-Experiments on Sequential Machines. In: SHANNON, Claude E. (Hrsg.) ; MACCARTHY, J. (Hrsg.): *Automata Studies*, Princeton University Press, 1956, S. 129–153

[MRLA82] MUDGE, T. N. ; RUTENBAR, R. A. ; LOUGHEED, R. M. ; ATKINS, D. E.: Cellular Image Processing Techniques for VLSI Circuit Layout Validation and Routing. In: *DAC '82: Proceedings of the 19th Design Automation Conference*. Piscataway, NJ, USA : IEEE Press, 1982, S. 537–543

[MS91] MAES, Christian ; SHLOSMAN, Senya B.: Ergodicity of Probabilistic Cellular Automata: A Constructive Criterion. In: *Communications in Mathematical Physics* 135 (1991), Nr. 2, S. 233–251

[MS08] MÜLLER-SCHLOER, C. ; SICK, B.: Controlled Emergence and Self-Organization. In: WÜRTZ, R. P. (Hrsg.): *Organic Computing, Understanding Complex Systems*. Springer, 2008, Kapitel 4, S. 81–103

[MSPP02] MESOT, Bertrand ; SANCHEZ, Eduardo ; PEÑA, Carlos-Andres ; PEREZ-URIBE, Andres: SOS++: Finding Smart Behaviors Using Learning and Evolution. In: STANDISH, R. (Hrsg.) ; BEDAU, M. (Hrsg.) ; ABBASS, H. (Hrsg.): *Artificial Life VIII: The 8th International Conference on Artificial Life*. Cambridge, Massachusetts : MIT Press, 2002, S. 264–273

[Mül96] MÜLLER, Jörg P.: *Lecture Notes in Artificial Intelligence and Lecture Notes in Computer Science*. Bd. 1177: *The Design of Intelligent Agents: A Layered Approach*. Springer, 1996

[MZG01] MARTÍNEZ BARBERÁ, Humberto ; ZAMORA IZQUIERDO, Miguel ; GÓMEZ SKARMETA, Antonio: Fuzzy Behaviours for Mobile Robots Control. In: *UK Workshop on Computational Intelligence (UKCI), Edinburgh*, 2001

[Neu66] NEUMANN, John von ; BURKS, Arthur W. (Hrsg.): *Theory of Self-Reproducing Automata*. Urbana, Illinois : University of Illinois Press, 1966

[NGB+09] NOUYAN, Shervin ; GROSS, Roderich ; BONANI, Michael ; MONDADA, Francesco ; DORIGO, Marco: Teamwork in Self-Organized Robot Colonies. In: *Trans. Evol. Comp* 13 (2009), Nr. 4, S. 695–711

[NKK96] NAUCK, Detlef ; KLAWONN, Frank ; KRUSE, Rudolf: *Neuronale Netze und Fuzzy-Systeme*. 2. Auflage. Wiesbaden : Vieweg, 1996

[NS92] NAGEL, K. ; SCHRECKENBERG, M.: A Cellular Automaton Model for Freeway Traffic. In: *J. de Physique* 2 (1992), S. 2221

[OC003] *VDE/ITG/GI-Positionspapier. Organic Computing - Computer- und Systemarchitektur im Jahr 2010*. http://www.betriebssysteme.org/Betriebssysteme/FutureTrends/oc-positionspapier.pdf, 2003

[OFM07] Olfati-Saber, Reza ; Fax, J. A. ; Murray, Richard M.: Consensus and Cooperation in Networked Multi-Agent Systems. In: *Proceedings of the IEEE* Bd. 95, IEEE, 2007, S. 215–233

[OL09] Otetelisanu, Joana ; Lindner, Marcus: *Untersuchung der Effizienz von Bewegungs- und Kommunikationsfähigkeiten für Agenten in Multi-Agenten-Systemen*. Studienarbeit, Technische Universität Darmstadt, 2009

[OM07] O'Donoghue, Diarmuid P. ; Mullally, Emma-Claire: Extending Irregular Cellular Automata with Geometric Proportional Analogies. In: *Geographical Information Science Research UK (GISRUK) Conference, NUI Maynooth, Ireland*, 2007, S. 353–358

[OT00] O'Sullivan, David ; Torrens, Paul M.: Cellular Models of Urban Systems. In: Bandini, Stefania (Hrsg.) ; Worsch, Thomas (Hrsg.): *Theoretical and Practical Issues on Cellular Automata, Proceedings of the Fourth International Conference on Cellular Automata for Research and Industry, Karlsruhe, 4-6 October 2000*, Springer, 2000, S. 108–116

[Pet06] Petrovic, Pavel: Comparing Finite State Automata Representation with GP-trees / Norwegian University of Science and Technology. 2006 (04/06). – IDI Technical Report

[PW02] Parsons, Simon ; Wooldridge, Michael: Game Theory and Decision Theory in Multi-Agent Systems. In: *Autonomous Agents and Multi-Agent Systems* 5 (2002), Nr. 3, S. 243–254

[Rag93] Raghavan, Raghu: Cellular Automata in Pattern Recognition. In: *Inf. Sci.* 70 (1993), Nr. 1-2, S. 145–177

[RBA96] Roisenberg, Mauro ; Barreto, Jorge M. ; Azevedo, Fernando M.: Biological Inspirations in Neural Network Implementations of Autonomous Agents. In: Borges, Díbio L. (Hrsg.) ; Kaestner, Celso A. A. (Hrsg.): *Advances in Artificial Intelligence, 13th Brazilian Symposium on Artificial Intelligence, SBIA '96, Curitiba, Brazil, October 23-25, 1996, Proceedings* Bd. 1159, Springer, 1996 (Lecture Notes in Computer Science), S. 211–220

[RDHK99] Rasek, Arno ; Dörwald, Walter ; Hauhs, Michael ; Kastner-Maresch, Alois: Species Formation in Evolving Finite State Machines. In: Floreano, Dario (Hrsg.) ; Nicoud, Jean-Daniel (Hrsg.) ; Mondada, Francesco (Hrsg.): *Advances in Artificial Life, 5th European Conference, ECAL'99, Lausanne, Switzerland, September 13-17, 1999, Proceedings* Bd. 1674, Springer, 1999 (Lecture Notes in Computer Science), S. 139–148

[Rey87] Reynolds, Craig W.: Flocks, Herds and Schools: A Distributed Behavioral Model. In: *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1987, S. 25–34

[RG85] Rosenschein, Jeffrey S. ; Genesereth, Michael R.: Deals Among Rational Agents. In: *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,*

Los Angeles, CA, 1985, S. 91–99. – (Auch veröffentlicht in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 227–234, Morgan Kaufmann, 1988.)

[RMS91] RITTER, Helge ; MARTINETZ, Thomas ; SCHULTEN, Klaus: *Neuronale Netze*. 2. Auflage. Addison-Wesley, 1991

[RN04] RUSSELL, Stuart J. ; NORVIG, Peter: *Künstliche Intelligenz: Ein moderner Ansatz*. 2. Auflage. München : Pearson Education Deutschland, 2004

[Ros95] ROSENBERG, Grant: *The Outer Limits Episode 01x17: The New Breed*. MGM Studios, TV Series, July 1995

[Ros05] ROSIN, Paul L.: Training Cellular Automata for Image Processing. In: KÄLVIÄINEN, Heikki (Hrsg.) ; PARKKINEN, Jussi (Hrsg.) ; KAARNA, Arto (Hrsg.): *Image Analysis, 14th Scandinavian Conference, SCIA 2005, Joensuu, Finland, June 19-22, 2005, Proceedings* Bd. 3540, Springer, 2005 (Lecture Notes in Computer Science), 195–204

[RPS08] RICHTER, Urban ; PROTHMANN, Holger ; SCHMECK, Hartmut: Improving XCS Performance by Distribution. In: LI, Xiaodong (Hrsg.) ; KIRLEY, Michael (Hrsg.) ; ZHANG, Mengjie (Hrsg.) ; GREEN, David G. (Hrsg.) ; CIESIELSKI, Victor (Hrsg.) ; ABBASS, Hussein A. (Hrsg.) ; MICHALEWICZ, Zbigniew (Hrsg.) ; HENDTLASS, Tim (Hrsg.) ; DEB, Kalyanmoy (Hrsg.) ; TAN, Kay C. (Hrsg.) ; BRANKE, Jürgen (Hrsg.) ; SHI, Yuhui (Hrsg.): *Simulated Evolution and Learning, 7th International Conference, SEAL 2008, Melbourne, Australia, December 7-10, 2008. Proceedings* Bd. 5361, Springer, 2008 (Lecture Notes in Computer Science), 111–120

[SAML08] SAMSUDIN, Khairulmizam ; AHMAD, Faisul A. ; MASHOHOR, Syamsiah ; LATIF, Norfadzilah M.: Comparison of Direct and Incremental Genetic Algorithm for Optimization of Ordinal Fuzzy Controllers. In: *SNPD*, IEEE Computer Society, 2008, 128–134

[Sch81] SCHULZ, Hans-Detlef: *Zellularautomaten zur Bearbeitung der "Lokal-Global-Problematik"*, Technische Hochschule Darmstadt, Diss., 1981

[Sch93] SCHMIDHUBER, Jürgen: *Netzwerkarchitekturen, Zielfunktionen und Kettenregel*. 1993. – Habilitationsschrift, Technische Universität München

[Sch04] SCHÄTZING, Frank: *Der Schwarm*. Kiepenhauer & Witsch, 2004

[Sch05] SCHMECK, Hartmut: Organic Computing - A New Vision for Distributed Embedded Systems. In: *ISORC*, IEEE Computer Society, 2005, 201–203

[Sch08] SCHADSCHNEIDER, Andreas: Conflicts and Friction in Pedestrian Dynamics. In: UMEO, Hiroshi (Hrsg.) ; MORISHITA, Shin (Hrsg.) ; NISHINARI, Katsuhiro (Hrsg.) ; KOMATSUZAKI, Toshihiko (Hrsg.) ; BANDINI, Stefania (Hrsg.): *ACRI* Bd. 5191, Springer, 2008 (Lecture Notes in Computer Science), 559–562

[SFS09] SPICHER, Antoine ; FATÈS, Nazim ; SIMONIN, Olivier: From Reactive Multi-Agents Models to Cellular Automata - Illustration on a Diffusion-Limited Aggregation Model. In: FILIPE, Joaquim (Hrsg.) ; FRED, Ana L. N. (Hrsg.) ; SHARP, Bernadette (Hrsg.): *ICAART*, INSTICC Press, 2009, S. 422–429

[SG00a] Spears, William M. ; Gordon, Diana F.: Evolution of Strategies for Resource Protection Problems. In: *Advances in Evolutionary Computing: Theory and Applications*. New York, NY, USA : Springer-Verlag, 2000, 367–392

[SG00b] Spears, William M. ; Gordon, Diana F.: Evolving Finite-State Machine Strategies for Protecting Resources. In: Ras, Zbigniew W. (Hrsg.) ; Ohsuga, Setsuo (Hrsg.): *Foundations of Intelligent Systems, 12th International Symposium, ISMIS 2000, Charlotte, NC, USA, October 11-14, 2000, Proceedings* Bd. 1932, Springer, 2000 (Lecture Notes in Computer Science), 166–175

[SHH10] Schäck, Christian ; Heenes, Wolfgang ; Hoffmann, Rolf: Efficient Traffic Simulation Using the GCA Model. In: *12th Workshop on Advances in Parallel and Distributed Computational Models (APDCM)*, 2010

[Sip94] Sipper, Moshe: Non-Uniform Cellular Automata: Evolution in Rule Space and Formation of Complex Structures. In: Brooks, Rodney A. (Hrsg.) ; Maes, Pattie (Hrsg.): *Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems ArtificialLifeIV*. Cambridge, MA, USA : MIT Press, Juli 1994, S. 394–399

[Sip97] Sipper, Moshe: *Lecture Notes in Computer Science*. Bd. 1194: *Evolution of Parallel Cellular Machines, The Cellular Programming Approach*. Springer, 1997

[SMVB05] Sauter, John A. ; Matthews, Robert S. ; Van Dyke Parunak, H. ; Brueckner, Sven: Performance of Digital Pheromones for Swarming Vehicle Control. In: Dignum, Frank (Hrsg.) ; Dignum, Virginia (Hrsg.) ; Koenig, Sven (Hrsg.) ; Kraus, Sarit (Hrsg.) ; Singh, Munindar P. (Hrsg.) ; Wooldridge, Michael (Hrsg.): *4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands*, ACM, 2005, 903–910

[Sou78] Soukup, J.: Fast Maze Router. In: *DAC '78: Proceedings of the 15th Design Automation Conference*. Piscataway, NJ, USA : IEEE Press, 1978, 100–102

[SS93] Schadschneider, A. ; Schreckenberg, M.: Cellular Automaton Models and Traffic Flow. In: *J. Phys A* 26 (1993), S. L679–L683

[SSCJ98] Shehory, O. ; Sycara, K. ; Chalasani, P. ; Jha, S.: Agent Cloning: An Approach to Agent Mobility and Resource Allocation. In: *IEEE Communications Magazine* 36 (1998), Juli, Nr. 7, 58–67

[ST99] Sipper, Moshe ; Tomassini, Marco: Computation in Artificially Evolved, Non-uniform Cellular Automata. In: *Theor. Comput. Sci.* 217 (1999), Nr. 1, S. 81–98

[THRR04] Truszkowski, Walt ; Hinchey, Mike ; Rash, James L. ; Rouff, Christopher: NASA's Swarm Missions: The Challenge of Building Autonomous Software. In: *IT Professional* 6 (2004), Nr. 5, S. 47–52

[TJA08] Tavakoli, Yashar ; Javadi, H. Haj S. ; Adabi, Sepideh: A Cellular Automata Based Algorithm for Path Planning in Multi-Agent Systems with a Common Goal. In: *IJCSNS, International Journal of Computer Science and Network Security* 8 (2008), Nr. 7, S. 119–123

[TM87] TOFFOLI, Tommaso ; MARGOLUS, Norman: *Cellular Automata Machines: A New Environment for Modeling*. Cambridge, MA, USA : MIT Press, 1987

[Tof84] TOFFOLI, Tommaso: CAM: A High - Performance Cellular - Automation Machine. In: *Physica D: Nonlinear Phenomena* 10 (1984), S. 195–204

[TSTJ05] TORKI, Samir ; SANZA, Cédric ; TORGUET, Patrice ; JESSEL, Jean-Pierre: Classifier System Based Autonomous Vehicles in HLA Distributed Driving Simulations. In: *International Conference on Computer Graphics and Artificial Intelligence (3IA), Limoges, 11/05/2005-12/05/2005*, Laboratoire XLIM - Université de Limoges, mai 2005, 149–159

[UMN+08] UMEO, Hiroshi (Hrsg.) ; MORISHITA, Shin (Hrsg.) ; NISHINARI, Katsuhiro (Hrsg.) ; KOMATSUZAKI, Toshihiko (Hrsg.) ; BANDINI, Stefania (Hrsg.): *Cellular Automata, 8th International Conference on Cellular Aotomata for Reseach and Industry, ACRI 2008, Yokohama, Japan, September 23-26, 2008. Proceedings*. Bd. *5191*. Springer, 2008 (Lecture Notes in Computer Science). – ISBN 978–3–540–79991–7

[Vol79] VOLLMAR, R.: *Algorithmen in Zellularautomaten*. Stuttgart : B.G. Teubner, 1979

[VVKB01] VALCKENAERS, Paul ; VAN BRUSSEL, Hendrik ; KOLLINGBAUM, Martin J. ; BOCHMANN, Olaf: Multi-agent Coordination and Control Using Stigmergy Applied to Manufacturing Control. In: LUCK, Michael (Hrsg.) ; MARÍK, Vladimír (Hrsg.) ; STEPÁNKOVÁ, Olga (Hrsg.) ; TRAPPL, Robert (Hrsg.): *Multi-Agent Systems and Applications, 9th ECCAI Advanced Course ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School, EASSS 2001, Prague, Czech Republic, July 2-13, 2001, Selected Tutorial Papers* Bd. 2086, Springer, 2001 (Lecture Notes in Computer Science), 317–334

[Wah00] WAHRIG, Gerhard: *Deutsches Wörterbuch*. Neu hrsg. von Renate Wahrig-Burfeind. Gütersloh/München : Bertelsmann, 2000

[War10] WARWICK, Kevin: Cultured Neural Networks. In: *Proceedings of the Institution of Mechanical Engineers* 224 (2010), Nr. 2, S. 109–111

[Wei99] WEISS, Gerhard (Hrsg.): *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA : The MIT Press, 1999

[WJ95] WOOLDRIDGE, M. ; JENNINGS, N. R.: Intelligent Agents: Theory and Practice. In: *Knowledge Engineering Review* 10 (1995), Nr. 2, S. 115–152

[WM99] WANG, Fang ; MCKENZIE, Eric: A Multi-Agent Based Evolutionary Artificial Neural Network for General Navigation in Unknown Environments. In: ETZIONI, Oren (Hrsg.) ; MÜLLER, Jörg P. (Hrsg.) ; BRADSHAW, Jeffrey M. (Hrsg.): *Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS-99)*, ACM Press, 1999, 154–159

[Wol83] WOLFRAM, Stephen: Cellular Automata. In: *Los Alamos Science* 9 (1983), S. 2–21

[Wol86a] WOLFRAM, Stephen: Cryptography with Cellular Automata. In: *Advances in Cryptology—CRYPTO '85 Proceedings* Bd. 218. New York, NY, USA : Springer-Verlag New York, Inc., 1986 (Lecture Notes in Computer Sciences), S. 429–432

[Wol86b] WOLFRAM, Stephen (Hrsg.): *Theory and Applications of Cellular Automata*. Singapore : World Scientific, 1986

[Woo02] WOOLDRIDGE, Michael: *An Introduction to MultiAgent Systems*. Chichester : Wiley, 2002

[WOO07] WEYNS, Danny ; OMICINI, Andrea ; ODELL, James: Environment as a First Class Abstraction in Multiagent Systems. In: *Autonomous Agents and Multi-Agent Systems* 14 (2007), Nr. 1, 5–30

[Wür08] *Kapitel* 1. In: WÜRTZ, R. P.: *Introduction: Organic Computing*. Springer, 2008, S. 1–6

[YHM+07] YAMAKAGE, Susumu ; HOSHIRO, Hiroyuki ; MITSUTSUJI, Katsuma ; SAKAMOTO, Takuto ; SUZUKI, Kazutoshi ; YAMAMOTO, Kazuja: Political Science and Multi-Agent Simulation: Affinities, Examples and Possibilities. In: TERANO, T. (Hrsg.) ; KITA, H. (Hrsg.) ; DEGUCHI, H. (Hrsg.) ; KIJIMA, K. (Hrsg.): *Agent-Based Approaches in Economic and Social Complex Systems IV (ABSS), Post Proceedings of The AESCS International Workshop 2005* Bd. 3, 2007 (Springer Series on Agent Based Social Systems)

[Zec02] ZECK, Günther: *Halbleiterchip mit einfachem biologischen neuronelen Netz*, Technische Universität München, Diss., 2002

Wissenschaftlicher Werdegang

Berufstätigkeit	Seit 09/2007	wissenschaftlicher Mitarbeiter an der Technischen Universität Darmstadt Fachgebiet: Rechnerarchitektur
	03/2007 - 08/2007	wissenschaftliche Hilfskraft mit Abschluss an der Technischen Universität Darmstadt Fachgebiet: Rechnerarchitektur
	04/2002 - 02/2004 und 04/2005 - 01/2006	wissenschaftliche Hilfskraft an der Technischen Universität Darmstadt Fachgebiete: Rechnerarchitektur, Integrierte Schaltungen und Systeme und Graphisch Interaktive Systeme
Hochschulausbildung	10/2000 - 02/2007	Studium der Informatik an der Technischen Universität Darmstadt Abschluss: Diplom-Informatiker
Wehrdienst	09/1999 - 06/2000	Artillerieregiment 13 Mühlhausen/Thüringen
Schulausbildung	07/1990 - 06/1999	Gymnasium Elly-Heuss-Schule Wiesbaden Abschluss: Abitur